УДК 370 + 378.1 + 681.142

ПРОВЕДЕННЯ ОБЧИСЛЮВАЛЬНОГО ЕКСПЕРИМЕНТУ ЗАСОБАМИ СИСТЕМИ ДИСТАНЦІЙНОГО ВИВЧЕННЯ КУРСУ «ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ»

Співаковський О.В., Осипова Н.В., Львов М.С., Бакуменко К.В.
Херсонський державний університет

У статті представлено короткий огляд можливостей інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування» (http://weboap.ksu.ks.ua), розробленого лабораторією інтегрованих середовищ навчання НДІ ІТ Херсонського державного університету. У рамках висвітлення теми «Обчислювальний експеримент» надано приклади розв'язання та аналізу ефективності алгоритмів сортування для масивів даних різної довжини, наочне представлення їх роботи засобами модулю «Середовище демонстрації», а також подано кілька варіантів визначення складності за часом виконання програми.

Ключові слова: інтегроване середовище, основи алгоритмізації та програмування, середовище демонстрації, обчислювальний експеримент, ефективність, алгоритм сортування, час виконання, масив вхідних даних

Постановка й обґрунтування актуальності проблеми.

Обчислювальний (або комп'ютерний) експеримент дозволяє студенту зрозуміти особливості певних алгоритмів та усвідомити залежності, що пояснюють їх складність. Обчислювальний експеримент складається з планування, створення або вибору експериментальної установки та виконання контрольних випробувань. За цим іде проведення серійних дослідів, обробка експериментальних даних та їх інтерпретація. Обчислювальний експеримент з вивчення ефективності алгоритмів проводиться за допомогою спеціально розробленого інтегрованого середовища. Таким чином можна говорити про обчислювальний експеримент як про нову технологію і методологію наукових і прикладних досліджень.

Обчислювальний експеримент – метод вивчення об'єктів та процесів за допомогою математичного моделювання. Він передбачає, що після побудови математичної моделі проводиться її чисельне дослідження, що дозволяє «програти» поведінку досліджуваного об'єкту в різних умовах або в різних модифікаціях.

Тож, важливим аспектом є розробка програмного забезпечення для обчислювального експерименту в конкретній області діяльності. Найкращим варіантом є створення крупного програмного комплексу, що складається із зв'язаних між собою прикладних програм і системних засобів. Вони у свою чергу повинні надавати користувачеві можливості управління ходом обчислювального експерименту, обробки і представлення його результатів [8].

Виклад основного матеріалу.

Сучасні комп'ютерні програми володіють високою сервісністю і доброзичливим інтерфейсом, що дозволяє легко освоїти роботу з ними за короткий час.

При проведенні досліджень важливо пам'ятати, що обчислювальний експеримент має свої обмеження, які можуть привести до неефективних витрат часу і ресурсів, або навіть до отримання помилкових результатів.

Напевно жодна інша проблема не породила такої кількості різноманітних рішень, як задача сортування. Об'єктом дослідження даної статті є алгоритми сортування елементів масиву. Зрозуміло, що універсального алгоритму не існує, але завдяки можливості порівняння характеристик можна віднайти найоптимальніший та найкорисніший для конкретної ситуації [3].

Спершу потрібно визначити ті параметри, за якими буде проводитися аналіз кожного алгоритму. Отже, ми зупинимося на наступних.

Час сортування – основний параметр, що характеризує швидкодію алгоритму.

Пам'ять. Ряд алгоритмів вимагає виділення додаткової пам'яті під тимчасове зберігання даних. Під час оцінки використовуваної пам'яті не враховуватиметься місце, яке займає початковий масив і незалежні від вхідної послідовності витрати, наприклад, на зберігання коду програми.

Стійкість. Стійке сортування не змінює взаємного розташування рівних елементів. Така властивість може бути дуже корисною, якщо елементи складаються з декількох полів.

Природність поведінки – ефективність методу при обробці вже відсортованих або частково відсортованих даних.

Мажорованість. Загальне математичне визначення тут наводити немає сенсу, та для теми обчислювального експерименту це поняття можна визначити у наступному контексті: при виконанні алгоритму на необмеженому масиві вхідних даних існує мажоранта - функція *f(n)* (власне кажучи, *n* - це кількість елементів масиву) така, що її значення не менші за значення вихідної функції у відповідних точках. Тож у даній статті ми намагатимемося для кожного алгоритму знайти таку мажоранту *f(n)* [6].

Для оцінки продуктивності алгоритмів можна використовувати різні підходи. Оцінити час виконання через символ *O(n)* (читається так: *O* велике від *n*).

O(g(n)) – множина функцій *f(n)*, для яких існують позитивні константи *c*, n_0, такі що *f(n) <= c*g(n)* для всіх $n>=n_0$.

При аналітичному підході складність алгоритмів доводиться за допомогою точних суворих викладок. Проте на практиці, студенти дуже важко сприймають цей матеріал.

З огляду на це ми пропонуємо використовувати інший спосіб: запустити кожен алгоритм на декількох масивах даних і дослідити вказані характеристики (визначити кількість порівнянь, кількість пересилань та час виконання), виявити масиви, для яких той чи інший алгоритм буде більш ефективним. Досить цікавим дослідженням для студентів є виявлення кількості елементів масиву, починаючи з якої один алгоритм має переваги над іншим.

Створення ефективного програмного забезпечення дозволяє швидко проводити експерименти та багато разів змінювати параметри [7].

У лабораторії інтегрованих середовищ навчання НДІ ІТ Херсонського державного університету розробленого інтегроване середовище вивчення курсу "Основи алгоритмізації та програмування" для вищих навчальних закладів, що призначено для використання у лекційно-аудиторній, дистанційній та заочній формах навчання для студентів педагогічних, технічних та економічних напрямків. Основними перевагами середовища є можливість організації за його допомогою самостійної роботи та поточного і підсумкового контролю знань студентів, а також організації обчислювального експерименту. Середовище надає як викладачу, так і студентам усі можливості ефективного вивчення курсу з основ алгоритмізації та програмування [5].

Користувачами програмного засобу є учні старших класів загальноосвітніх навчальних закладів, студенти вищих навчальних закладів, які вивчають основи алгоритмізації і програмування, вчителі з інформатики, викладачі.

Робочою мовою програмування в курсі основ алгоритмізації та програмування обрана навчальна мова програмування Pascal.

Клас задач інтегрованого середовища курсу «Основи алгоритмізації та програмування» – алгоритми обробки масивів даних, у тому числі сортування, пошук унікальних елементів (максимуми, мінімуми тощо). Утім, використання програми не обмежено лише цим класом задач.

Одним з модулів системи дистанційного навчання є середовище демонстрації. Воно призначене для наочної демонстрації роботи алгоритмів на лекціях, при проведенні практичних занять і лабораторних робіт. Використання даного модулю дозволяє більше

уваги приділити саме аналізу алгоритмів: на різних масивах даних в результаті виконання демонстрації визначаються основні характеристики – кількість порівнянь та кількість перестановок.

В середовищі демонстрації користувач має можливість вибрати і відкрити алгоритм для демонстрації з колекції системи або з колекції користувача, ініціювати дані демонстрації, налагодити демонстрацію, виконати демонстрацію в неперервному або покроковому режимах.

Використання динамічних образів операцій присвоювання, порівняння, передачі параметрів в процедури та функції, рекурсивних викликів процедур та функцій, процесу генерації вхідних даних робить середовище демонстрації виключно корисним засобом вивчення основ алгоритмізації.

Таким чином, завдяки можливостям середовища викладач має змогу урізноманітнити види практичних завдань з алгоритмізації [2]:

- виконати алгоритм з колекції системи або колекції користувача для певних даних;
- скласти алгоритм розв’язання задачі;
- визначити ефективність алгоритму;
- порівняти ефективність алгоритмів для певного набору даних;
- дослідити та змоделювати дані для певного алгоритму (випадковим чином, найкращий та найгірший випадки та ін.);
- узагальнити результати аналізу алгоритмів при порівнянні різних методів розв’язання задачі;
- запропонувати більш ефективний алгоритм розв’язання задачі.

Будемо використовувати середовище демонстрації для оцінки складності та ефективності роботи алгоритмів сортування.

Спочатку розглянемо процес аналізу ефективності одного з алгоритмів, а саме сортування вибором.

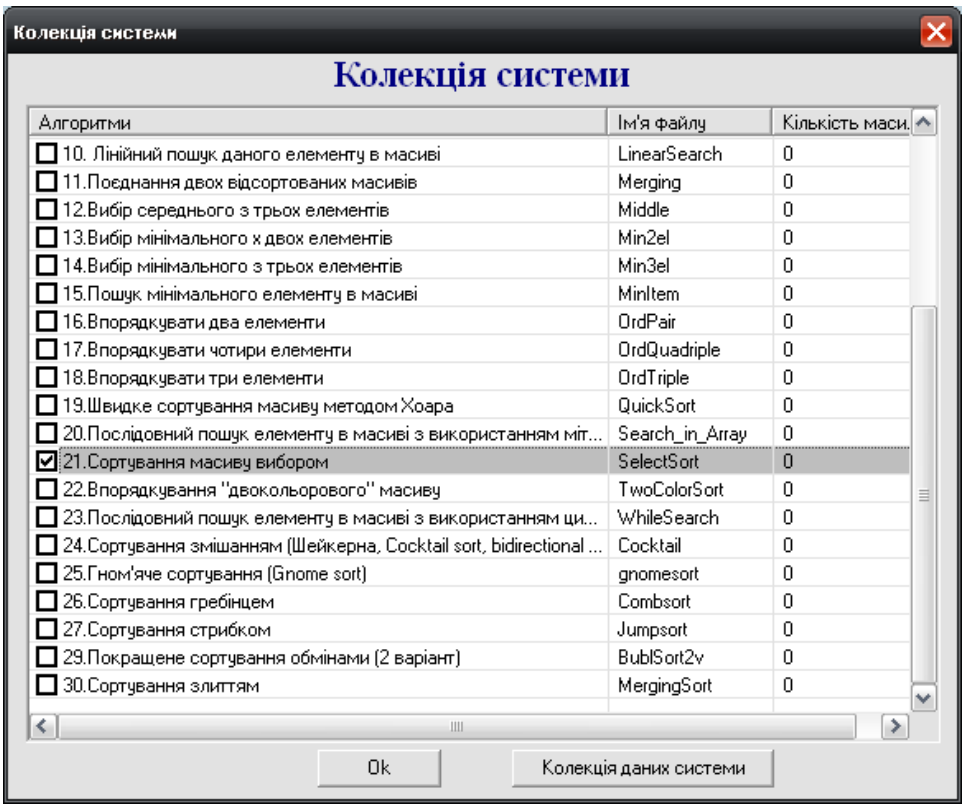
Постановка задачі. Дано масив *A* довжиною 10 елементів. Відсортувати його за зростанням методом сортування вибором. Визначити складність алгоритму за кількістю порівнянь та перестановок.

Для розв’язання даної задачі можна скористатися колекцією системи та обрати у ній відповідний алгоритм (мал. 1).

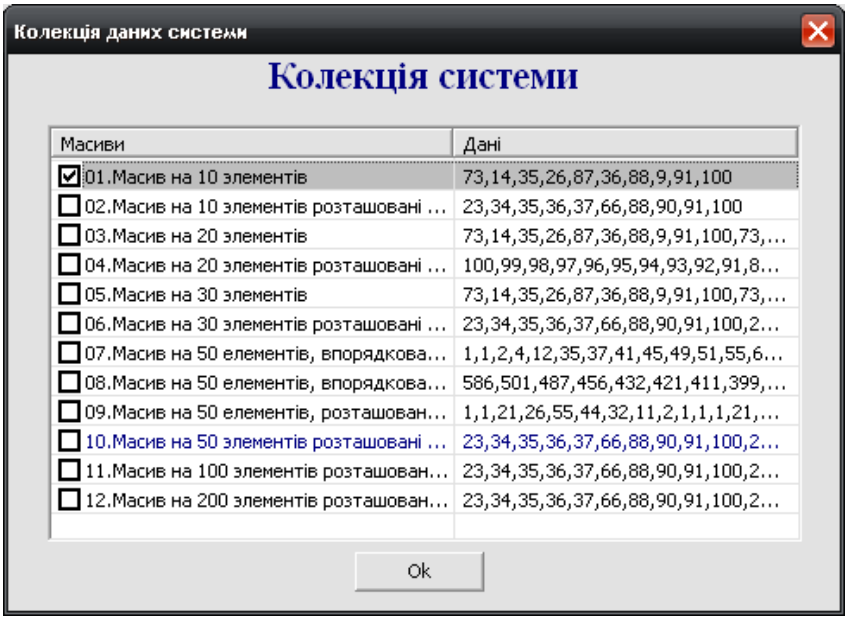
У колекції даних системи необхідно обрати масив з 10 елементів (мал.2).

Після цього алгоритм відобразиться у лівій частині вікна.

```
Program SelectSort;
Const
  n = 10;
Var
  a : array[1..n] of Data;
  i, j, MinInd : Integer;
  b : Data;
Begin
  For i := 1 to n - 1 do begin
    MinInd := i;
    For j := i + 1 to n do
      If a[j] < a[MinInd]
       then MinInd := j;
    b := a[MinInd];
    a[MinInd] := a[i];
    a[i] := b
  end;
End.
```



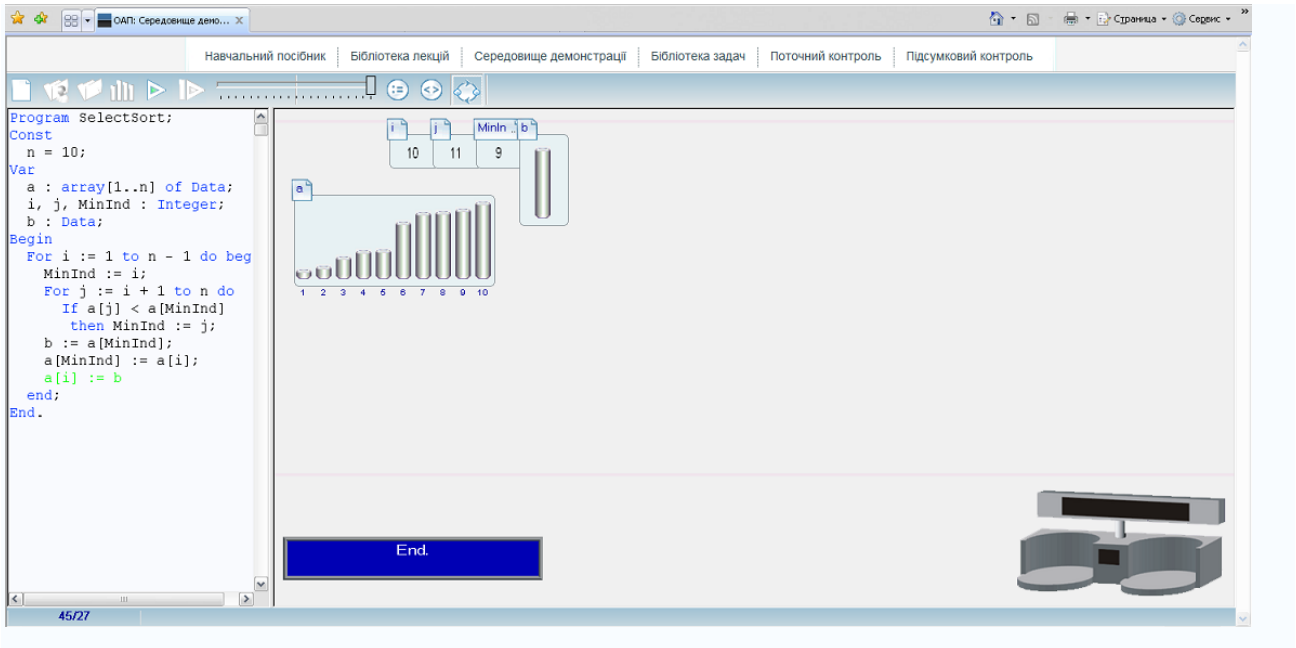
Мал. 1. Колекція алгоритмів системи



Мал. 2. Колекція даних системи

При натисненні на кнопку «*Заповнити даними*» (), для автоматичного заповнення необхідно перенести змінні-аргументи з лівої частини діалогового вікна *«Формування даних»* до правої та обрати «*Застосувати*». Після того, як дані сформовано, стане активною кнопка «*Запустити*» (). Після її натиснення на екрані буде відображатися наочне виконання алгоритму (мал. 3).

Для визначення складності алгоритму за кількістю порівнянь та пересилань, необхідно записати кількість порівнянь та кількість пересилань за кожен лінійний перегляд, а також кількість лінійних проходів.



Мал. 3. Модуль середовище демонстрації

Після виконання алгоритму у лівому нижньому куті екрану відображається загальна кількість порівнянь та пересилань (45/27).

В результаті дослідження для масиву з кількістю елементів 10 отримано наступні дані:

Кількість порівнянь за кожен лінійний перегляд – 9, 8, 7, 6, 5, 4, 3, 2, 1.

Кількість пересилань за кожен лінійний перегляд – 1.

Кількість лінійних переглядів – 9.

Числовий ряд кількості порівнянь представляє собою арифметичну прогресію від 1 до 9, тому, побачивши цю закономірність, можна скористатися формулою суми $S(n)=n(n-1)/2$, де $n=9$. В результаті отримаємо відповідь 45. Отже, складність алгоритму методом вибору для кількості порівнянь дорівнює $O(n^2)$.

Для визначення складності даного алгоритму за кількістю пересилань необхідно визначити їх загальну кількість, тобто 9*1=9. Крім того, врахуємо, що кожне пересилання включає три операції присвоєння, тому отримане значення необхідно помножити на 3. Таким чином отримано число 27, яке відповідає показникам з середовища демонстрації.

Можна відмітити залежність кількості елементів масиву n від кількості проходів та виразити це математичною формулою: $n-1$. Тепер можна вивести власне формулу для кількості пересилань: $3*(n-1)$. Тоді складність алгоритму за цим параметром складе $O(n)$.

Аналітично виразити дані формули можна наступним чином.

Необхідно позначити складність алгоритму сортування масиву $a[1..n]$ за кількістю порівнянь $C(n)$, а за кількістю пересилань – $M(n)$.

Основна дія сортування вибором – пошук найменшого елемента в частині масиву, що переглядається, і перестановка з першим елементом цієї частини [1]:

For i := 1 To n – 1 Do
Begin
k := Індекс(Min(a[i], ..., a[n]));
Переставити a[i], a[k]
End;

Внутрішній цикл здійснює пошук мінімального елемента. Після виконання оператора *If* має місце співвідношення

$$Min = Min(a[i], a[i+1], \dots , a[j]),$$

а після завершення циклу

$$Min = Min(a[i], a[i+1], \dots , a[n]).$$

Після перестановки маємо

$$a[i] = Min(a[i], a[i+1], \dots , a[n]).$$

Зовнішній цикл керує довжиною частини масиву, що переглядається. Після виконання тіла зовнішнього циклу початок масиву вже відсортований:

$$a[1] <= a[2] <= .... <= a[i].$$

Після завершення зовнішнього циклу отримаємо:

$$a[1] <= a[2] <= ... <= a[n-1], a[n-1] = Min(a[n-1], a[n]),$$

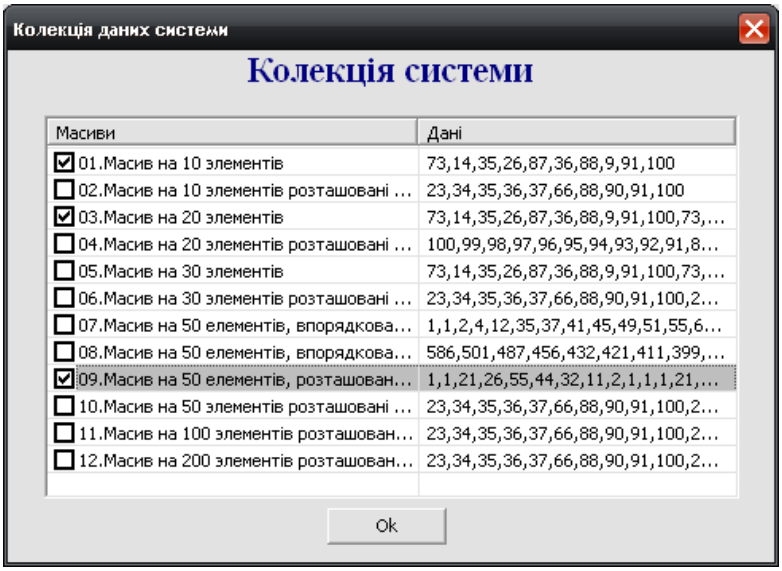
тобто масив відсортований.

Зовнішній цикл виконався $n-1$ разів. Внутрішній цикл виконується $i-1$ разів ($i = n-1, n-2, ..., 1$). Кожне виконання тіла внутрішнього циклу полягає в одному порівнянні. Тому $C(n) = 1 + 2 + ... + n - 1 = n(n - 1)/2$.

Перестановка елементів здійснюється у зовнішньому циклі. Тому $M(n) = 3(n - 1)$.

Отже, дані, отримані практичним шляхом, легко доводяться аналітично.

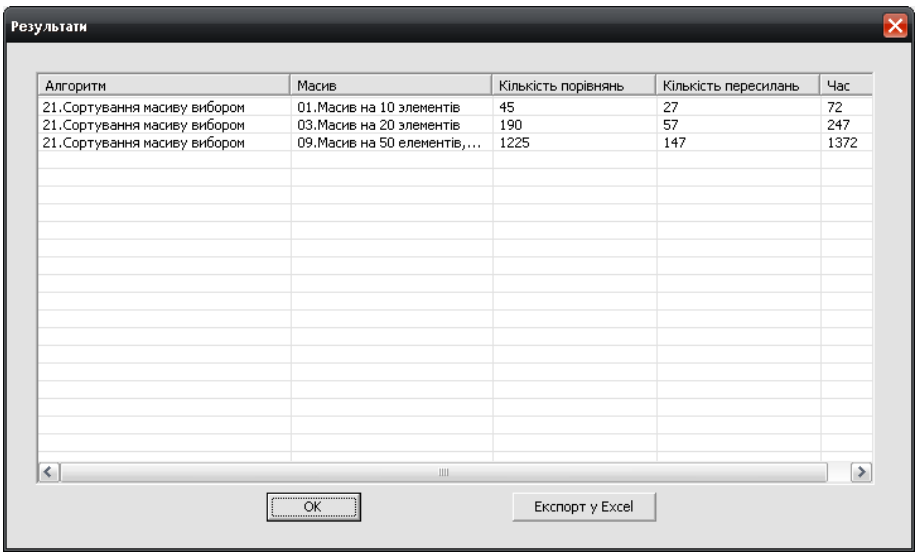
Крім того, для візуального представлення результатів можна виконати алгоритм на кількох масивах даних з різною кількістю елементів. Для цього в колекції даних системи необхідно вибрати не один, а кілька масивів, як це показано на малюнку 4:



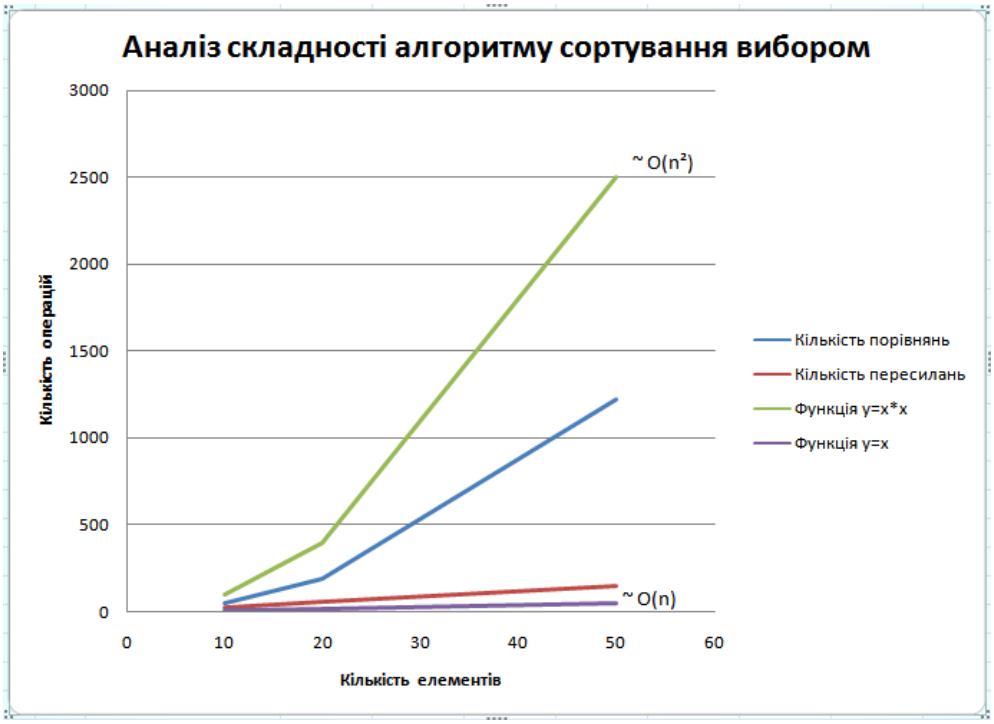
Мал. 4. Вибір кількох масивів у колекції даних системи

Після завершення демонстрації виконання алгоритму на трьох масивах даних отримаємо наступні результати (мал.5), які можна експортувати в Excel для подальшої обробки та графічної інтерпретації.

За отриманими даними можна побудувати графіки залежності кількості операцій (порівнянь та пересилань) від розмірності масиву вхідних даних (мал. 6). Крім того, до діаграми слід додати також графіки функцій $y=x^2$ та $y=x$. У такому разі дійсно видно, що кількість пересилань пропорційна до функції $y=x$, а кількість порівнянь – до $y=x^2$ [4].



Мал. 5. Результати виконання алгоритму на кількох масивах даних



Мал. 6. Аналіз складності алгоритму сортування вибором

Крім того, засобами середовища демонстрації можна оцінити ефективність роботи алгоритмів сортування відносно один одного навіть на великих масивах вхідних даних. Тож, далі для кількісного аналізу візьмемо наступні алгоритми колекції системи у середовищі демонстрації:

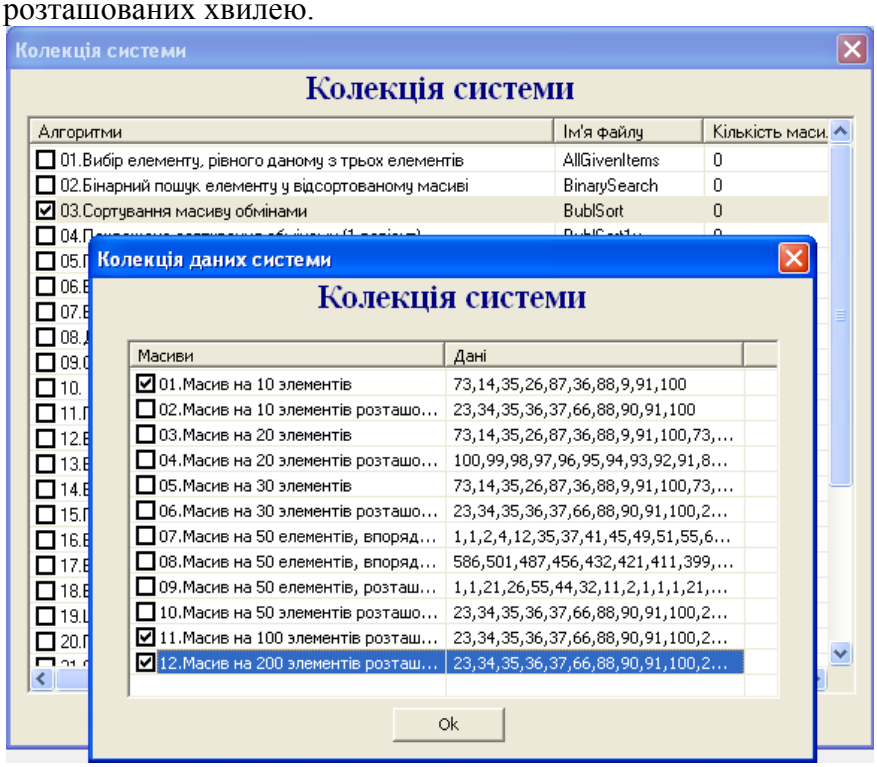
- Сортування масиву обмінами.
- Покращене сортування обмінами (1 варіант).
- Сортування масиву методом вставки.
- Швидке сортування масиву методом Хоара.
- Сортування масиву вибором.
- Покращене сортування обмінами (2 варіант).
- Сортування злиттям.

Проведемо моделювання роботи кожного з них на масивах різної довжини. При цьому проаналізуємо ефективність та швидкодію кожного з алгоритмів.

Для порівняння алгоритмів були використані масиви довжиною в 10, 100 та 200 елементів.

Отже, перейдемо до покрокового виконання алгоритмів сортування масиву. Необхідно увійти до середовища демонстрації інтегрованого середовища WebОАП.

Потрібно натиснути на кнопку «Відкрити колекцію». У діалоговому вікні, що з'явилося (мал. 7), обрати перший вид сортування з наведеного вище переліку (у колекції – 03.Сортування масиву обмінами), натиснути на кнопку «Колекція даних» і обрати 3 пункти: 01.Масив на 10 елементів, 11.Масив на 100 елементів, розташованих хвилею та 12.Масив на 200 елементів, розташованих хвилею.



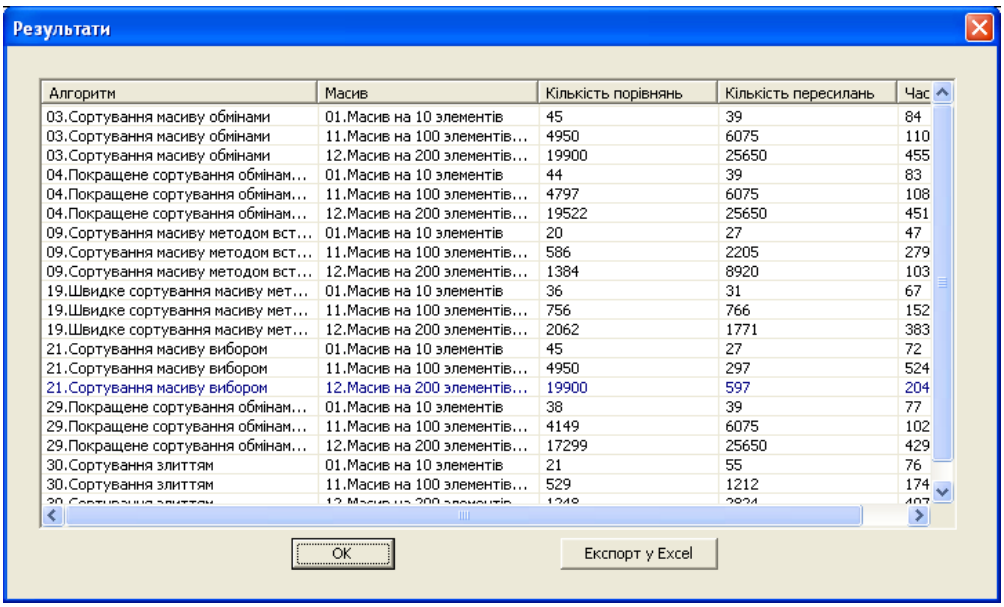
Мал. 7. Вибір алгоритмів та даних з колекції

Після цього натиснути кнопку «ОК» і, не виходячи з колекції системи, повторити вибір алгоритму та даних для кожного виду сортувань.

Після натиснення кнопки «ОК» у вікні колекції системи перший з обраних алгоритмів відобразиться у лівій частині екрану.

Після натиснення на кнопку «Заповнити даними» () необхідно обрати «Застосувати». Коли змінні заповняться, стане активною кнопка «Почати» (). Після її натиснення на екрані буде відображатися демонстрація виконання алгоритму. При виконанні алгоритму автоматично здійснюється підрахунок кількості порівнянь та пересилань у лівому нижньому куті екрану.

Щоб продовжити виконання всіх алгоритмів сортувань, необхідно знову натиснути на кнопку «Почати» (завантажиться текст алгоритму), потім на кнопку «Заповнити даними» (обрати «Застосувати») і натиснути на кнопку «Почати». Таким чином потрібно виконати всі обрані алгоритми, після чого натиснути на кнопку «Почати». На екрані з'явиться вікно з результатами, які потрібно експортувати до Excel (кнопка «Експорт у Excel») (мал. 8).



Алгоритм	Масив	Кількість порівнянь	Кількість пересилань	Час
03.Сортування масиву обмінами	01.Масив на 10 елементів	45	39	84
03.Сортування масиву обмінами	11.Масив на 100 елементів…	4950	6075	110
03.Сортування масиву обмінами	12.Масив на 200 елементів…	19900	25650	455
04.Покращене сортування обмінам…	01.Масив на 10 елементів	44	39	83
04.Покращене сортування обмінам…	11.Масив на 100 елементів…	4797	6075	108
04.Покращене сортування обмінам…	12.Масив на 200 елементів…	19522	25650	451
09.Сортування масиву методом вст…	01.Масив на 10 елементів	20	27	47
09.Сортування масиву методом вст…	11.Масив на 100 елементів…	586	2205	279
09.Сортування масиву методом вст…	12.Масив на 200 елементів…	1384	8920	103
19.Швидке сортування масиву мет…	01.Масив на 10 елементів	36	31	67
19.Швидке сортування масиву мет…	11.Масив на 100 елементів…	756	766	152
19.Швидке сортування масиву мет…	12.Масив на 200 елементів…	2062	1771	383
21.Сортування масиву вибором	01.Масив на 10 елементів	45	27	72
21.Сортування масиву вибором	11.Масив на 100 елементів…	4950	297	524
21.Сортування масиву вибором	12.Масив на 200 елементів…	19900	597	204
29.Покращене сортування обмінам…	01.Масив на 10 елементів	38	39	77
29.Покращене сортування обмінам…	11.Масив на 100 елементів…	4149	6075	102
29.Покращене сортування обмінам…	12.Масив на 200 елементів…	17299	25650	429
30.Сортування злиттям	01.Масив на 10 елементів	21	55	76
30.Сортування злиттям	11.Масив на 100 елементів…	529	1212	174

	A	B	C	D	E	F
1	Алгоритм	Масив	Кількість порівнянь	Кількість пересилань	Час	
2	03.Сортування масиву обмінами	01.Масив на 10 елементів	45	39	84	
3	03.Сортування масиву обмінами	11.Масив на 100 елементів розташовані хвилею	4950	6075	11025	
4	03.Сортування масиву обмінами	12.Масив на 200 елементів розташовані хвилею	19900	25650	45550	
5	04.Покращене сортування обмінами (1 варіант)	01.Масив на 10 елементів	44	39	83	
6	04.Покращене сортування обмінами (1 варіант)	11.Масив на 100 елементів розташовані хвилею	4797	6075	10872	
7	04.Покращене сортування обмінами (1 варіант)	12.Масив на 200 елементів розташовані хвилею	19522	25650	45172	
8	09.Сортування масиву методом вставки	01.Масив на 10 елементів	20	27	47	
9	09.Сортування масиву методом вставки	11.Масив на 100 елементів розташовані хвилею	586	2205	2791	
10	09.Сортування масиву методом вставки	12.Масив на 200 елементів розташовані хвилею	1384	8920	10304	
11	19.Швидке сортування масиву методом Хоара	01.Масив на 10 елементів	36	31	67	
12	19.Швидке сортування масиву методом Хоара	11.Масив на 100 елементів розташовані хвилею	756	766	1522	
13	19.Швидке сортування масиву методом Хоара	12.Масив на 200 елементів розташовані хвилею	2062	1771	3833	
14	21.Сортування масиву вибором	01.Масив на 10 елементів	45	27	72	
15	21.Сортування масиву вибором	11.Масив на 100 елементів розташовані хвилею	4950	297	5247	
16	21.Сортування масиву вибором	12.Масив на 200 елементів розташовані хвилею	19900	597	20497	
17	29.Покращене сортування обмінами (2 варіант)	01.Масив на 10 елементів	38	39	77	
18	29.Покращене сортування обмінами (2 варіант)	11.Масив на 100 елементів розташовані хвилею	4149	6075	10224	
19	29.Покращене сортування обмінами (2 варіант)	12.Масив на 200 елементів розташовані хвилею	17299	25650	42949	
20	30.Сортування злиттям	01.Масив на 10 елементів	21	55	76	
21	30.Сортування злиттям	11.Масив на 100 елементів розташовані хвилею	529	1212	1741	
22	30.Сортування злиттям	12.Масив на 200 елементів розташовані хвилею	1248	2824	4072	
23						

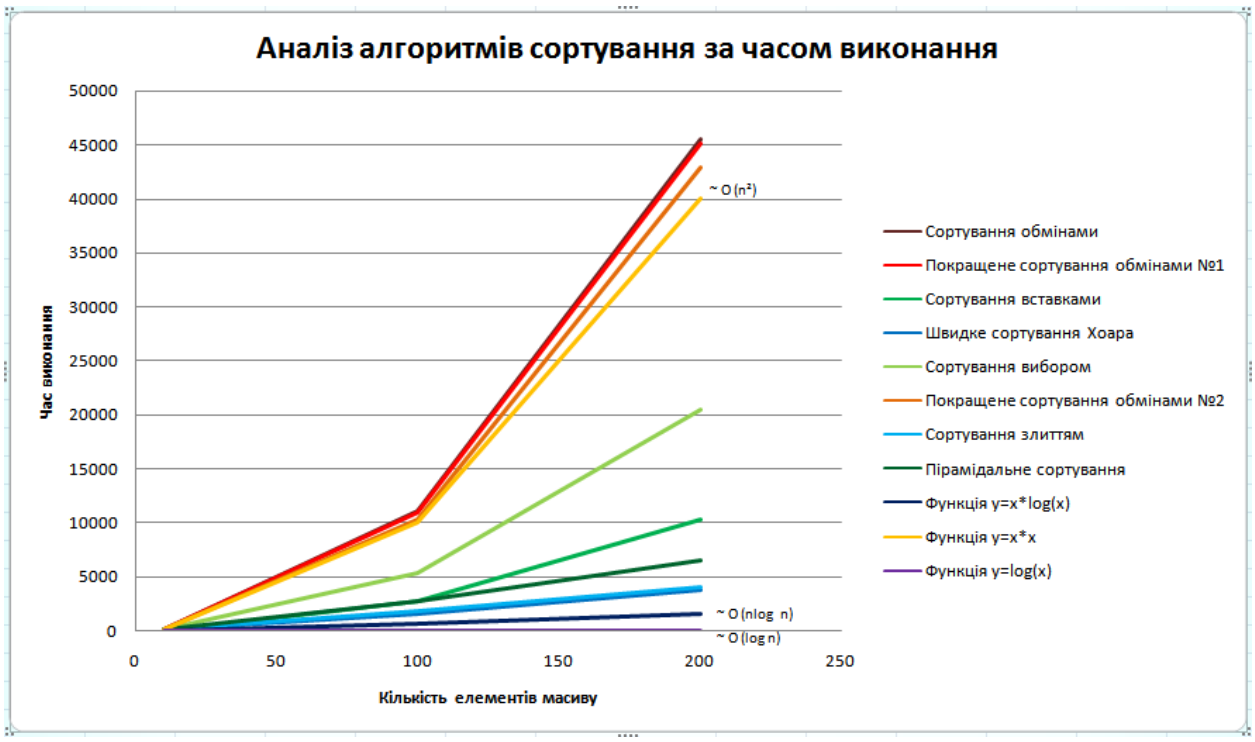
Мал. 8. Результати виконання алгоритмів

Проведемо аналіз алгоритмів за тими показниками, які були перераховані на початку статті.

Для першого параметру – *часу сортування* – представимо діаграму з графіками залежності часу виконання від кількості елементів масиву, якщо їх розмірність не перевищує 200 (мал. 9).

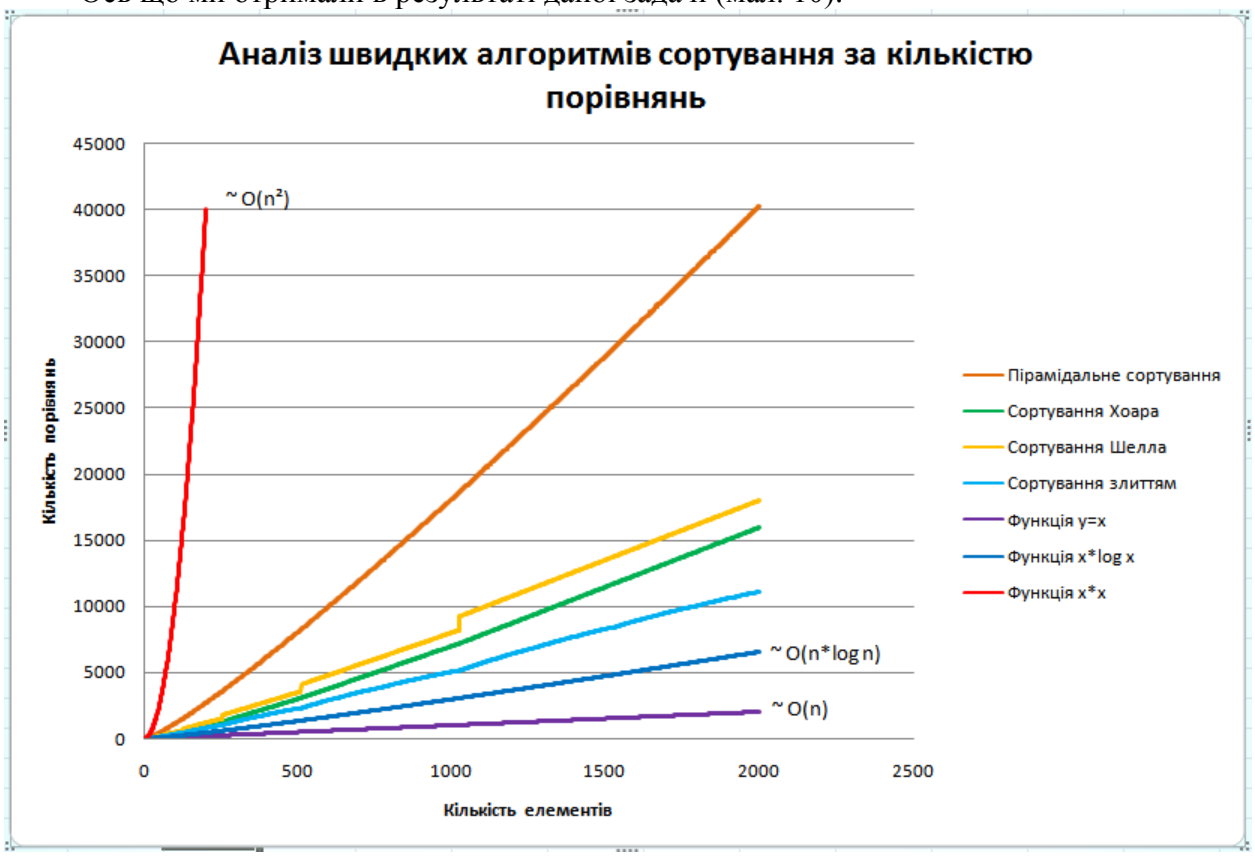
З аналізу графіків стає зрозумілим, яку складність має кожний алгоритм сортування за кількістю порівнянь.

Однак, в класичній літературі з програмування наголошується, що складні алгоритми сортувань розраховані на роботу з масивами великої розмірності, тому інколи при невиконанні цієї умови ми й отримуємо відповідь, що вони працюють гірше за прості сортування. Тож, для доведення даного факту проаналізуємо поведінку чотирьох алгоритмів – сортування Хоара, сортування Шела, сортування злиттям та пірамідальне сортування – на масивах довжиною до 2000 елементів.



Мал. 9. Аналіз алгоритмів сортування за часом виконання

Ось що ми отримали в результаті даної задачі (мал. 10):



Мал. 10. Аналіз швидких алгоритмів сортування за кількістю порівнянь

З діаграми видно, що алгоритми мають складність *O(n log n)*.

Другий критерій – *пам'ять*. Додатковий масив використовується лише в алгоритмі сортування злиттям. Інші ж алгоритми використовують лише одну додаткову змінну для пересилань елементів масиву.

Стійкість. Не змінили взаємного розташування елементів у вже відсортованому масиві даних сортування лише сортування обмінами, перший та другий його покращені варіанти, динамічне сортування та алгоритм Шела. Тож саме вони є стійкими і у разі використання масивів з елементами складної структури дозволять значно зекономити час сортування.

Природністю поведінки відзначилися алгоритми сортування обмінами, перший та другий його покращені варіації та алгоритм Шела. Вони є ефективними при обробці вже відсортованих даних.

Мажорованість. Використовуючи дані, представлені на рисунку, можна знайти таку функцію мажоранту f(x). Наприклад, для пірамідального сортування константа *c* (відношення кількості порівнянь до функції *y=x*log(x)* для відповідної кількості елементів) наближено дорівнює 6,3, а для сортування Хоара – 2,9. Тепер для знаходження мажоранти необхідно помножити цю константу *c* на значення функції *y=x*log(x)* для відповідних кількостей елементів та побудувати графіки. Наприклад, для пірамідального сортування вони будуть мати наступний вигляд (мал. 11):

Мал. 11. Знаходження мажоранти для пірамідального сортування

Отже, мажорантою для пірамідального сортування за кількістю порівнянь буде функція y=6,3*x*log(x).

Таким чином, нам вдалося практичним шляхом довести твердження Кнута про те, що «при N=1000 значення середнього часу виконання дорівнюють приблизно

160000u для пірамідального сортування,

130000u для сортування методом Шела,

80000u для швидкого сортування (Хоара)»

Мова тут не йде про числові показники, проте чітко видно, що відносність розташування та ефективності алгоритмів сортування співпадає з наведеним зауваженням.

Висновки.

Проаналізувавши результати проведених експериментів, можна сказати, що кожен з алгоритмів веде себе по-різному на різних масивах даних.

Обчислювальний експеримент дозволяє відкрити нові властивості досліджуваних процесів. Саме в цьому і полягає його головна перевага.

Переваги обчислювального експерименту очевидні. В обчислювальний експеримент можна легко і безпечно втручатися. Його можна повторити і перервати у будь-який момент. В ході експерименту можна змоделювати необхідні умови.

Відомо, що застосовність результатів обчислювального експерименту обмежена рамками прийнятої математичної моделі. Звичайно, обчислювальний експеримент не може повністю замінити аналітичні доведення. Тож майбутнє за їх розумним поєднанням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кнут Д. Искусство программирования, том 3. Сортировка и поиск – М.: Вильямс, 2007. – 824 с.
2. Колеснікова Н.В., Надєєва А.В. Система демонстрації програм та контролю знань в інтегрованому середовищі вивчення курсу “Основи алгоритмізації та програмування”// Інформаційні технології в освіті: Збірник наукових праць. Випуск 1.– Херсон: Видавництво ХДУ, 2008.– С. 55-59.
3. Львов М.С., Спиваковский А.В., Белоусова С.В. Основы программирования на языке Паскаль. Херсон: МИБ, 1997.– 153 с.
4. Самарский А.А., Михайлов А.П. Математическое моделирование: Идеи. Методы. Примеры. – 2-е изд., испр. – М.: Физматлит, 2002. – 320с.
5. Спиваковский А.В., Гудырева Е.М., Кравцов Г.М. Технологии дистанционного образования как елементы, компенсирующие сокращение аудиторной нагрузки студента //Матер. Міжн. наук. - пр. конф. “Інформатизація освіти України: стан, проблеми, перспективи” – Херсон, 2001. – С. 22-24
6. Спиваковский А.В., Колесникова Н.В., Ткачук Н.И., Ткачук И.М. Web-среда для изучения основ алгоритмизации и программирования // Управляющие системы и машины.– Киев, 2008.– С. 70-75.
7. Співаковський О.В, Колеснікова Н.В. Відеоінтерпретатор алгоритмів інтегрованого середовища вивчення курсу “Основи алгоритмізації та програмування”// Збірник праць Третьої Міжнародної конференції "Нові інформаційні технології в освіті для всіх: система електронної освіти".– Київ, 2008.– С. 399-404.
8. Співаковський О.В. Про вплив інформаційних технологій на технології освіти //Матер. Міжн. наук. -пр. конф. “Інформатизація освіти України: стан, проблеми, перспективи” – Херсон, 2001. – С. 129-131

Рецензент: Раков С.А.