УДК 004.415.3
Пех П.А.,Бортник К.Я., Яручик.О. В.
Луцький національний технічний університет

ПРОГРАМНИЙ С++КОМПЛЕКС ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ШВИДКОДІЇ АЛГОРИТМІВ СОРТУВАННЯ МЕТОДОМ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ

Пех П.А.,Бортник К.Я., Яручик О. В. Програмний С++комплекс та результати дослідження швидкодії алгоритмів сортування методом імітаційного моделювання. В статті запропоновано програмний комплекс засобами середовища C++Builder для дослідження швидкодії алгоритмів сортування методом імітаційного моделювання. Наведені результати дослідження швидкодії різних алгоритмів сортування.
Ключові слова: C++Builder проект, алгоритми сортування, швидкодія алгоритму, імітаційна модель, генератор випадкових чисел

Пех П.А., Бортник Е.Я., Яручик А. В. Программный С++комплекс и результаты исследования быстродействия алгоритмов сортировки методом имитационного моделирования. В статье предложен программный комплекс средствами C++Builder для исследовани быстродействия алгоритмов сортировки методом имитационного моделирования. Приведены также результаты исследования быстродействия различных алгоритмов сортировки.
Ключевые слова: C++Builder проект, алгоритмы сортировки, быстродействие алгоритма, имитационная модель, генератор случайных чисел

Pekh Petro, Bortnyk K.Ya., Yaruchyk Oleksandr. The C++program complex and the results of the sorting algorithms speed study by the simulation method. The article proposes C ++ Builder program complex to investigate the sorting algorithms speed using the simulation method. Also the investigating results of the sorting speed various algorithms are given.
Keywords: C ++ Builder project, sorting algorithms, algorithm speed, simulation model, random number generator

Постановка задачі. Швидкодія алгоритму сортування масиву є важливим параметром, з допомогою якого оцінюють ефективність цього алгоритму[1-5]. У більшості випадків її визначають математичним шляхом за середнім числом операцій порівняння та обмінів, які потрібно виконати для досягнення мети. Однак на час сортування масивів істотньо впливають і багато інших факторів, які можуть виявитися не менш важливими, а інколи і визначальними. Серед них – тип процесора; завантаженість процесора; кількість процесорів та можливість їх паралельної роботи; тип даних елементів масиву, що сортуються; програмне середовище, у якому реалізується алгоритм сортування, та інші. Саме тому швидкодія алгоритмів сортування є предметом цього дослідження, а вирішувати цю багатофакторну проблему ми пропонуємо методами імітаційного моделювання.

Важливим параметром є також потреба в додатковій пам'яті. Маються на увазі потреби в пам'яті під додаткові змінні та масиви, нарощення кеш-пам'яті та інші. Крім цих двох основних параметрів, інколи увага приділяється питанням стійкості алгоритму та його детермінованості, причому стійким вважається алгоритм сортування, який не змінює порядку розташування елементів вихідного масиву з однаковими індексами.

Метою нашого дослідження було розроблення засобами середовища `C++Builder Code Gear` сучасного програмного комплексу для імітаційного моделювання процесу сортування одновимірного масиву та визначення за його допомогою часу сортування у разі використання різних алгоритмів. Для дослідження нами вибрані одинадцять відомих алгоритмів сортування [6-11]. Новизна полягає саме у комплексному підході до вирішення проблеми та використанні методів імітаційного моделювання.

Основна частина. Планом передбачалося проведення досліджень у три етапи:

— розроблення засобами середовища `C++Builder` програмного комплексу для імітаційного моделювання процесу сортування одновимірного масиву;
— перевірка адекватності імітаційної моделі;
— проведення досліджень на створеній імітаційній моделі з використанням конкретних алгоритмів сортування.

Основою програмного комплексу є реалізований на консолі середовища `C++Builder` комплекс програм імітаційного моделювання процесів сортування одновимірного масиву. Кожна з цих програм реалізує один з одинадцяти методів сортування у вигляді окремого консольного додатка:

1. `bubble -` сортування методом бульбашки (обмінами);
2. `insertion -` сортування методом вставки;

3. `merge` - сортування методом злиття;
4. `cocktail` - сортування методом перемішуванням;
5. `gnome` - сортування методом гнома;
6. `tree` - сортування методом дерева;
7. `selection` - сортування методом вибору мінімального елемента;
8. `shell` - сортування методом Шелла;
9. `quick`- швидке сортування;
10. `stoog` – рекурсивне сортування;
11. `heap` - сортування методом кучію

Для того, щоб зрозуміти логіку побудови комплексу, наведемо коди лише кожної з перших п'яти програм. Програмний код першого `C++Builder` проекту наведено повністю.

```
//01_bubble - Сортування масиву методом бульбашки (обмінами)
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <windows.h>
#include <ctime.h>
using namespace std;
int i, k;
int nom;
int mmin,h;
const int a_size = 10000; //Обсяг одновимірного масиву a
int a[a_size];            //Оголошення одновимірного масиву a
unsigned int start_time,  //Момент часу початку сортування
             end_time,    //Момент часу закінчення сортування
             search_time; //Тривалість процесу сортування

int main() {
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);

  srand(time(0)); //Запуск генератора випадкових чисел

  //Формування однієї вибірки (масиву) випадкових чисел
  for (i = 0; i < a_size; i++) {
    //Формування поточного випадкового числа a[i] в діапазоні [0;100]
    a[i] = rand()% 100;  }

  // Друк елементів сформованого масиву a[a_size]:
  cout<<"\n Сформований масив до впорядкування: \n";
  for (i=0; i<a_size; i++)
   cout<<setw(4)<<a[i]<<" ";
  cout<<"\n";
  cout<<"\n Після аналізу даних натисніть клавішу Enter";
  cout<<"\n та почекайте на вивід впорядкованого масиву.";
  getch();

  //Запам'ятовуємо момент часу початку процесу впорядкування
  start_time = clock();
  //Реалізація алгоритму впорядкування масиву a методом бульбашки:
  // Поки не реалізовано a_size-1 кроків, виконуйте
  for (k=1; k<a_size; k++) {
    //Поки не переглянуто всіх елементів, виконуйте
    for (i=0; i<a_size-1; i++)
```

```
      if (a[i]>a[i+1]) {
        h=a[i];
        a[i]=a[i+1];
        a[i+1]=h;
      }
  }
  //Кінець реалізації алгоритму впорядкування масиву методом бульбашки

  //Запам'ятовуємо момент часу закінчення процесу впорядування
  end_time = clock();

  //Визначаємо тривалість процесу впорядкування
  search_time = end_time - start_time;

  // Друк елементів впорядкованого масиву a:
  cout<<"\n  Заданий масив після впорядкування:"<<"\n";
  for (i=0; i<a_size; i++)
    cout<<setw(4)<<a[i]<<" ";
  cout<<"\n";
  cout<<"\n Після аналізу даних натисніть клавішу Enter";
  getch();

  //Статистичні параметри оброблення масиву:
  cout<<" Обсяг масиву  a_size = " << a_size << endl;
  cout<<" Час початку процесу впорядкування start_time = "
      << start_time<<endl;
  cout<<" Час закінчення процесу впорядкування end_time = "
      <<end_time<<endl;
  cout<<" Тривалість процесу впорядкування search_time = "
      << search_time << endl;
  getch();
  return 0;
}
```

Як бачимо, перший програмний `C++Builder` проект забезпечує сортування одновимірного масиву методом бульбашки (обмінів). Зупинимось на цій програмі детальніше. У розділі директив компілятору та оголошень змінних оголошено: одновимірний масив `a[a_size]`цілого типу розміром `a_size` елементів, моменти часу `start_time` та `end_time` відповідно початку та кінця процесу сортування і його тривалості `search_time`. Значення макроконстанти `a_size` задається командою `const int a_size = 10000`.

У головній програмі здійснюється підключення функцій `SetConsoleCP(1251)`та `SetConsoleOutputCP(1251)`бібліотеки `<windows.h>`, які забезпечують виведення результатів моделювання українською мовою. Далі запускається генератор випадкових чисел `srand(time(0))`, використовуючи який та функцію `rand()`формуємо у циклі вибірку (одновимірний масив) цілих чисел заданого обсягу. Перед сортуванням фіксуємо момент часу початку сортування.

Серцевиною програми є та його частина, яка впорядковує одновимірний масив відповідним методом, у даному випадку – методом бульбашки. Зазначимо, що це один із найвідоміших методів сортування, а тому не будемо зупинятись на ньому більш детально. Впорядкувавши масив, фіксуємо момент часу закінчення процесу сортування і обчислюємо тривалість процесу сортування. Насамкінець, виводимо на екран усі вхідні та вихідні параметри процесу сортування.

Оскільки вибірка цілих чисел складається з випадкових чисел, які генеруються комп'ютером, то для отримання достовірних результатів необхідно кожного разу проводити серію таких імітаційних експериментів, тобто ставити багатофакторний кібернетичний експеримент.

Для всіх наступних програм наводимо тільки ті їх частини, які є їх серцевинами.

```
//02_insertion - сортування масиву методом вставки
  // Впорядкування одновимірного масиву a методом вставки:
  for (k=1; k<a_size; k++) {
    b=a[k];
    j=0;
    while ((b>a[j]) && (j<k))
     j++;
    if (j<k){
      b=a[k];
      for(i=k; i>=j+1; i--)
        a[i]=a[i-1];
      a[j] = b;
    }
  } //Кінець реалізації алгоритму впорядкування масиву методом вставки

//03_merge - сортування масиву методом злиття
  void merge(int l, int r);
  int main() {

  // Впорядкування масиву методом злиття:
  merge(0, a_size - 1);

  return 0;
}
//Функція, що реалізує метод злиття
void merge(int l, int r) {
  if (r == l)
    return;
  if (r - l == 1) {
    if (a[r] < a[l])
      swap(a[r], a[l]);
    return;
  }
  int m = (r + l) / 2;
  merge(l, m);
  merge(m + 1, r);
  int buf[a_size];
  int xl = l;
  int xr = m + 1;
  int cur = 0;
  while (r - l + 1 != cur) {
    if (xl > m)
      buf[cur++] = a[xr++];
    else if (xr > r)
      buf[cur++] = a[xl++];
    else if (a[xl] > a[xr])
      buf[cur++] = a[xr++];
    else buf[cur++] = a[xl++];
  }
  for (i = 0; i < cur; i++)
    a[i + l] = buf[i];
}

//04_cocktail - сортування перемішуванням. Інші назви методу:
//(shaker - шейкерне, shuttle - трансфертне, ripple - пульсуюче)
//функція обміну значень елементів a[i-i] та a[i]
```

```
void Swap(int *Mas, int i);
//функція шейкерного сортування масиву, що містить елементи
//масиву Mas зі Start - го по N - ий
void ShakerSort(int *Mas, int Start, int N);
int main() {

  //Реалізація алгоритму впорядкування масиву a методом перемішування:
  ShakerSort(a, 1, a_size);

  return 0;
}

//функція обміну значень елементів a[i-i] та a[i]
void Swap(int *Mas, int i) {
 int temp;
 temp=Mas[i];
 Mas[i]=Mas[i-1];
 Mas[i-1]=temp;
}

//функція шейкерного сортування масиву, що містить елементи масиву Mas
//зі Start - го по N - ий
void ShakerSort(int *Mas, int Start, int N) {
 int Left, Right, i;
 Left=Start;
 Right=N-1;
 while (Left<=Right) {
  for (i=Right; i>=Left; i--)
   if (Mas[i-1]>Mas[i]) Swap(Mas, i);
  Left++;
  for (i=Left; i<=Right; i++)
   if (Mas[i-1]>Mas[i]) Swap(Mas, i);
  Right--;
 }
}

//05_gnome - сортування методом гнома
void Gnome(int a[], int N);
int main() {

// Реалізація алгоритму впорядкування масиву a методом гнома:
  Gnome(a, a_size);
  //Кінець реалізації алгоритму впорядкування

  return 0;
}
// Реалізація сортування методом гнома
void Gnome(int a[], int N) {
  int i, tmp;
  while (i<N) { // поки не переглянули всіх елементів масиву:
  //за відсутності попереднього елемента
  //зразу переходимо до наступного елемента:
  if (i==0) i=1 ;
  //якщо попередній елемент менший або дорівнює наступному,
  //переходимо до наступного:
  if (a[i-1]<=a[i]) i++;
```

```
  else { // в протилежному випадку міняємо елементи місцями
    tmp=a[i];
    a[i]=a[i-1];
    a[i-1]=tmp;
    i--;  //і повертаємося на крок назад
  }
   }
}
```

Окремі результати імітаційного моделювання у вигляді графіків залежностей часу сортування масиву від обсягу вибірки наведено на рис.1. Час сортування масиву вимірювався у тактах таймера комп'ютера. Обсяг вибірки задавали як значення макроконстанти.

Як бачимо з цих графіків, найбільш ефективним з наведених п'яти алгоритмів, виявився алгоритм швидкого сортування, а найменш ефективним з точки зору його швидкодії виявився все той же широко відомий метод бульбашки.

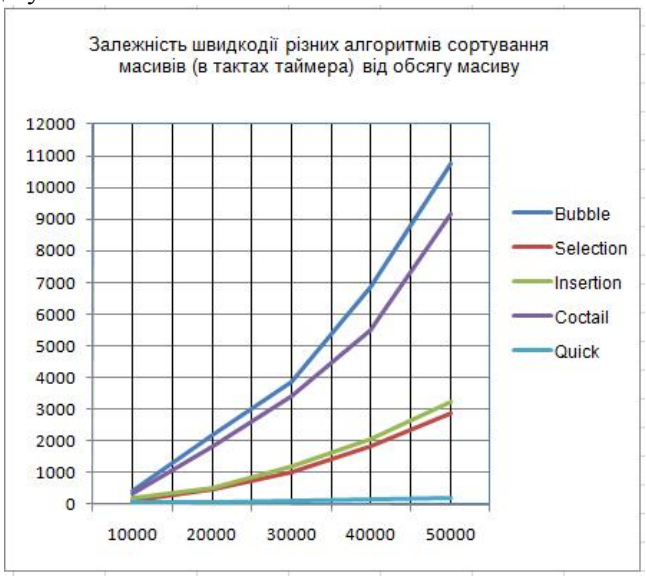

Рис. 1. Залежності часу сортування масиву різними алгоритмами від обсягу вибірки

Висновок. В статті розроблено C++Builder програмний комплекс, який дозволяє досліджувати швидкодію різних алгоритмів сортування методом імітаційного моделювання. Наведені результати дослідження деяких з цих алгоритмів.

1. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поискю -The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
2. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = Introduction to algorithms. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 5-8459-0857-4.
3. Роберт Седжвик. Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. — СПб.: ДиаСофтЮП, 2003. — С. 672. — ISBN 5-93772-081-4.
4. Дейтел Х.М., Дейтел П.Дж. Как программировать на С++. –М.: М.: ООО "Бином-Пресс", 2011.- 1456с.
5. Архангельский А.Я. Приемы программирования в С++ Builder 6 и 2006. – М.: ООО "Бином-Пресс", 2010.- 992с.
6. http://phys. bspu. by/static/lib/inf/prg/vb/vb6_1/glava1/gl1_5_2. htm
7. http://www.ipkro. isu.ru/informat/methods/findsort/sort. htm
8 http://algolist. manual.ru/sort/bubble_sort. php
9. http://algolist. manual.ru/sort/select_sort. php
10. http://algolist. manual.ru/sort/insert_sort. php
11. http://algolist. manual.ru/sort/quick_sort. php
12. http://algolist. manual.ru/sort/shell_sort. php