Центральноукраїнський національний технічний унiверситет

Кафедра програмування комп’ютерних систем і мереж

Дисципліна: Базові методології та технології програмування

Звiт

з лабораторної роботи №9-10

Тема: Технології програмування алгоритмів сортування та пошуку в масивах даних

Мета роботи: Отримання практичних навичок в обробці масивів, у сортуванні елементів масивів різними методами та за різними реквізитами; дослідження та вивчення методів пошуку ключових елементів у масивах; здійснення порівняння та аналізу ефективності використовуваних методів сортування та пошуку.

Виконав: ст. гр. КМ-20

Калиновський В.М.

Перевірила: асистент кафедри ПКСМ

Тесленко О.Є.

Варіант 3

Дата виконання: 30.11.2020

Дата оформлення: 01.12.2020

**Кропивницький 2020 р.**

**Завдання:** Згенерувати масив динамічно та вивести його на екран (використовувати при цьому власну функцію для виведення масиву), відсортувати масив, обравши вказані методи та параметри сортування; запускаючи програму не менше 10 разів для кожного методу (або передбачити це у програмі), задавати різну кількість *n* елементів масиву, отримати час *t* сортування, побудувати залежність *t=f(n)* на одному графіку для різних методів сортування у табличному і графічному виглядах; здійснити пошук вказаного з клавіатури елемента у масиві, використовуючи вказаний метод пошуку згідно варіанту. Варіант 3: для сортування та пошуку використовувати окремі функції; методи сортування: Шелла, вибору; Тип елементів: дійсні; Напрям сортування: з кінця; Порядок сортування: за убуванням; Метод пошуку: бінарний.

**Робота.** Складаю алгоритм моєї програми:

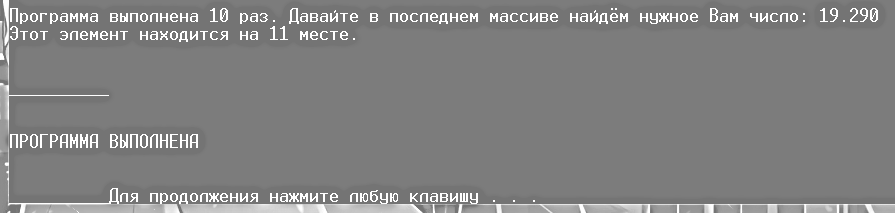
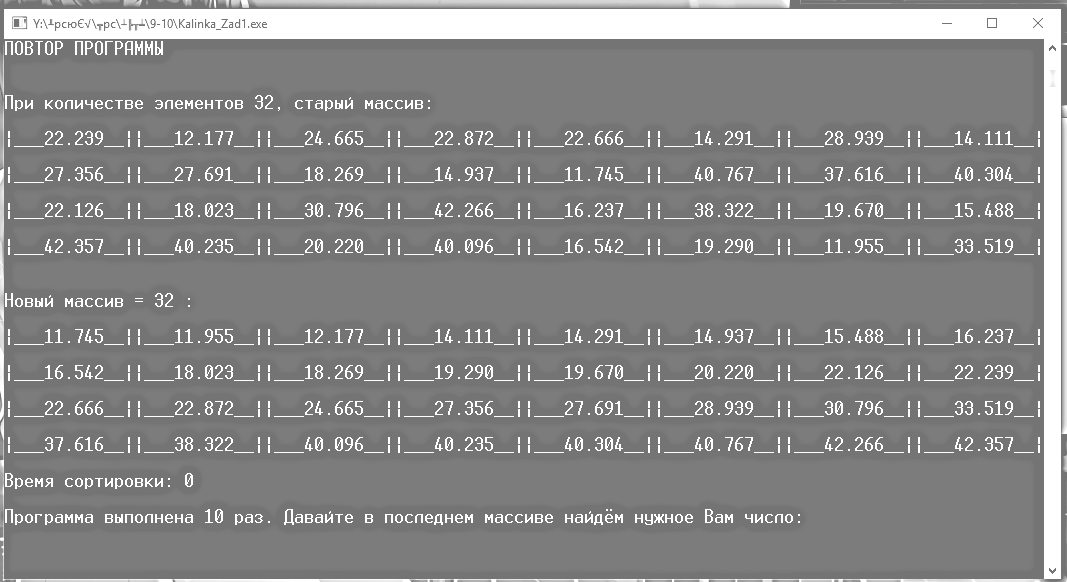
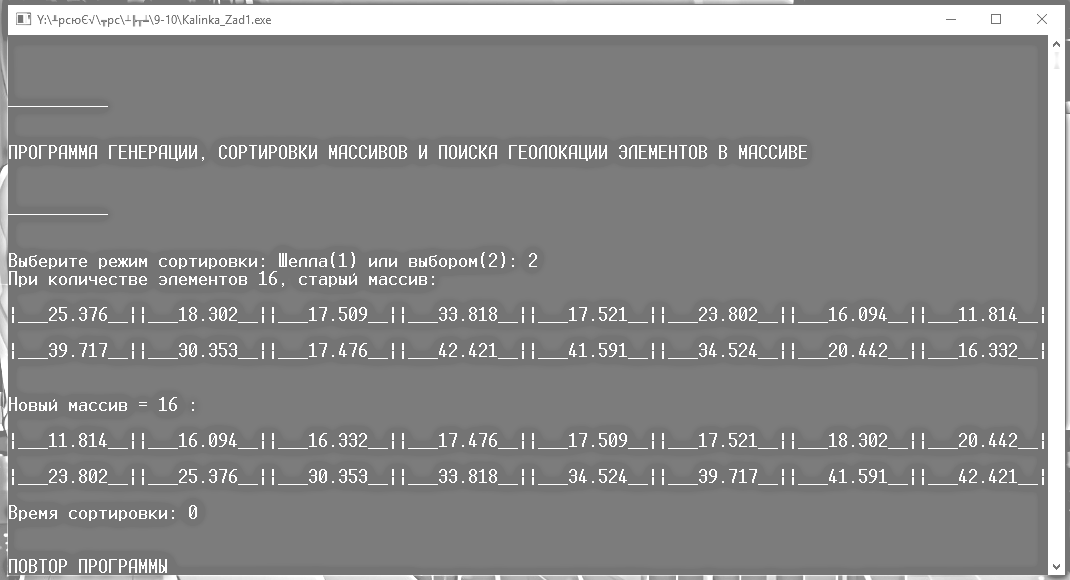
1. Програма питає, який метод користування хоче задіяти користувач: Шелла чи вибору
2. Програма рандомно генерує розмір масиву
3. В залежності від згенерованого розміру, програма генерує рандомно елементи, кількість яких дорівнює розміру масива
4. Програма виводить на екран згенерований масив
5. В залежності від обраного методу, програма сортує масив
6. Програма виводить відсортований масив
7. Потім рахує час затрачений на сортування
8. Пункти 2 – 7 програма виконує десять разів
9. Після у останньому масиві програма пропонує знайти номер необхідного елемента
10. У разі його присутності – програма виводить результат, а відсутності – -1

У таблиці 1 розписано код програми, а на рисунках 1, 2, 3 зображено результат програми. *Відразу відмічу, що я спеціально не став вкладати всі десять масивів, а вклав тільки перший і останній для економії чорнил та паперу, а також я вкоротив генерацію до розмірів «2, 4, 8, 16, 32», щоб вони вмістилися повністю на вікно виводу інформації. Насправді, програма може генерувати масиви розмірами «2, 4, 8, 16, 32, 64, 128, 256, 512 та 1024» елементів.*

Таблиця 1. Код програми генерації та сортування масивів

|  |
| --- |
| #include <Windows.h> //Нужна для подключения русского и функции sleep  #include <stdio.h> //Нужна для форматированого ввода-вывода  #include <stdlib.h> //Подключаю библиотеку с функцией рандома  #include <time.h> //Нужна для функции time и clock  //Функция вывода результата  void vyvod(float Massiv[], int Capacity) {  for (int p = 0; p < Capacity; p++) {  //Текущие восемь элементов выводятся в строку, а следующие сношу на следующую  if (p % 8 != 0){  printf("|\_\_\_%.3f\_\_|", Massiv[p]);  }  else {  printf("\n|\_\_\_%.3f\_\_|", Massiv[p]);  }  }  }  //Функция сортировки методом Шелла  void sort1(float Massiv[], int Capacity) {  unsigned int Start\_time = clock(); //Начальное время  int shag = Capacity;  while (shag > 0) {  for (int n = 0; n < Capacity - shag; n++) {  //Сравниваю элементы и выставляю меньший на первое место  float \*Menshee = &Massiv[n];  float \*Tekushee = &Massiv[n+shag];  if (\*Menshee > \*Tekushee) {  float Zamen = \*Menshee;  \*Menshee = \*Tekushee;  \*Tekushee = Zamen;  }  }  shag /= 2;  }  unsigned int End\_time = clock(); //Конечное время  unsigned int Search\_time = (End\_time - Start\_time); //Искомое время  printf("\n\nНовый массив = %i :\n", Capacity); //Начинаю выводить массив  vyvod(Massiv, Capacity); //Вывожу отсортированный массив  printf("\nВремя сортировки: %u", Search\_time); //Вывожу время сортировки  }  //Функция сортировки методом выбора  void sort2(float Massiv[], int Capacity) {  unsigned int Start\_time = clock(); //Начальное время    for (int Prosmotr = 0; Prosmotr < Capacity; Prosmotr++){  float \*Menshee = &Massiv[Prosmotr];  //Сравниваю первое число с каждым, и в конце первым должно стоять наименьшее  //На каждый следующий круг будет оставаться всё меньше элементов  for (int Prosm = Prosmotr + 1; Prosm < Capacity; Prosm++) {  float \*Tekushee = &Massiv[Prosm];  if (\*Menshee > \*Tekushee) {  float Zamen = \*Menshee;  \*Menshee = \*Tekushee;  \*Tekushee = Zamen;  }  }  }  unsigned int End\_time = clock(); //Конечное время  unsigned int Search\_time = (End\_time - Start\_time); //Искомое время  printf("\n\nНовый массив = %i :\n", Capacity); //Начинаю выводить массив  vyvod(Massiv, Capacity); //Вывожу отсортированный массив  printf("\nВремя сортировки: %u", Search\_time); //Вывожу время сортировки  }  //Функция поиска  int BinarySearch(float Massiv[], float SearchEl, int Left, int Right) {  if (Right >= Left) {  //Нахожу половину, в которой есть искомое число и каждый раз сужаю половину пополам  //В конце должен вывестись номер элемента или -1  int mid = (Right + Left) / 2;  if (Massiv[mid] == SearchEl) {  return mid;  }  if (Massiv[mid] > SearchEl) {  return BinarySearch(Massiv, SearchEl, Left, mid - 1);  }  if (Massiv[mid] < SearchEl) {  return BinarySearch(Massiv, SearchEl, mid + 1, Right);  }  }  return -1;  }  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);    printf("\n\n\n\_\_\_\_\_\_\_\_\_\_\n\n\nПРОГРАММА ГЕНЕРАЦИИ, СОРТИРОВКИ МАССИВОВ И ПОИСКА ГЕОЛОКАЦИИ ЭЛЕМЕНТОВ В МАССИВЕ\n\n\n\_\_\_\_\_\_\_\_\_\_\n\n\n");  int Capacity, Vybor\_sort, Vyzov = 0;  boolean Vyhod = 1;  //Спрашиваю, как хочет пользователь отсортировать массив  printf("Выберите режим сортировки: Шелла(1) или выбором(2): ");  do {  fflush(stdin);  scanf("%i", &Vybor\_sort);  switch (Vybor\_sort) {  case 1:  Vyhod = 0;  Vyzov = 1;  break;  case 2:  Vyhod = 0;  break;  default:  Vyhod = 1;  printf("Введено неправильное значение. Повторите попытку: ");  break;  }  } while (Vyhod == 1);  /\*Начинаю цикл, где будут генерироваться 10 раз массивы и сортироваться выбранным методом.  В конце цикл спрашивает, какой элемнт хочет найти пользователь и вызывает функцию поиска.\*/  for (int p = 0; p < 10; p++) {  srand(time(NULL)); //Беру текущее время для функции рандома  //Создаю рандомно массив, который будет состоять из 2, 4, 8, 16, 32, 64, 128, 256 или 512 элементов  Capacity = 10 + rand() % 1000;  for (int z = 2; z < Capacity \* 2; z = z \* 2) {  if (Capacity < z) {  Capacity = z;  }  }  float Massiv[Capacity];  //Генерирую для каждого элемента своё значение  for (int i = 0; i < Capacity; i++) {  float element = 10000 + rand() % 17342561;  Massiv[i] = element / 1000;  }  //Вывожу на экран начальный состав массива  printf("При количестве элементов %i, старый массив:\n", Capacity);  vyvod(Massiv, Capacity);  Sleep(500); //Ожидаю полсекунды перед сортировкой (измеряется в тысячных долях секунды  //Сортирую массив в зависимости от выбранного пользователем метода  if (Vyzov == 1) {  sort1(Massiv, Capacity);  }  else {  sort2(Massiv, Capacity);  }  if (p < 9) {  printf("\n\n\nПОВТОР ПРОГРАММЫ\n\n\n");  }  Sleep(1500); //Заставляю ждать программу полторы секунды  //Начинаю часть с сортировкой: пользователь вводит число, которое хочет найти...  if (p == 9) {  printf("\n\nПрограмма выполнена 10 раз. Давайте в последнем массиве найдём нужное Вам число: ");  float SearchEl;  fflush(stdin);  scanf("%f", &SearchEl);  //... и вызывается функция поиска местонахождения числа в массиве  int SearchInFunc = BinarySearch(Massiv, SearchEl, 0, Capacity - 1);  printf("Этот элемент находится на %i месте.", SearchInFunc); //Вывожу номер элемента  }  }  //Завершаю программу  printf("\n\n\n\_\_\_\_\_\_\_\_\_\_\n\n\nПРОГРАММА ВЫПОЛНЕНА\n\n\n\_\_\_\_\_\_\_\_\_\_");  system("pause");  return 0;  } |

Рисунки 1, 2, 3. Результат виконання програми табл.1



Також відмічу, чому час сортування завжди дорівнює 0: комп’ютер, на якому відбувалось сортування – достатньо потужний, тому він досить швидко сортує масиви. Дійсно, на масиви до 126 елементів він затрачує 0 секунд, на 256 – 1-2 секунди, на 512 – 5-6 секунд. Оскільки я навмисне скоротив генерацію масивів, то масиви генерувалися тільки ті, що сортуються швидше, ніж за одну секунду. Тому й такий результат.

Налагоджуючи масив, я виміряв час сортування для кожного типу сортування з будь-яким розміром і отримав залежність. З неї можна зрозуміти, що у певний момент метод сортування на графіку починає зростати. До 64 елементів вони затрачують час однаково – менше 1 секунди, а потім метод вибору починає потребувати більше часу. Це пов’язано з тим, що цей метод перевіряє кожен елемент з найменшим, а тому при збільшенні елементів затрачуваний час зростає експоненціально, тоді як метод Шелла використовує підмасивне сортування, де початку один масив ділиться на кількість масивів, що вдвічі менша від кількості елементі, кожен з яких має при собі всього два елементи; потім кількість масивів зменшується в два рази, а їх кількість елементів збільшується в два рази тощо, поки не залишиться один масив. Таким чином ми можемо зробити математичну модель. Метод вибору перший раз перевіряє усі елементи, наступного разу на один менше, наступного ще на один менше тощо – отримуємо факторіал кількості елементів (n!), а у методі Шелла перший раз сортуються усього два елементи, потім чотири, потім вісім тощо до останніх, що дорівнюють кількості елементів масиву. Останній крок візьмемо за n, так як у першому методі n дорівнювало загальній кількості елементів. Тому передостанній буде , перед-перед-останній дорівнює => отже їх загальна сума прямує до 2. І чим більша кількість елементів, тим ближче це число до двох, тоді як у методі вибору n=2 вже на другому етапі. Математично ці залежності можна описати так:

1. Вибору
2. Шелла

Робимо висновок, що метод вибору буде завжди більше потребувати часу, ніж Шелла, при чому чим більше елементів, тім більше експоненціально потрібно більше часу, але такий метод дуже точний і він ніколи не видає помилок, тоді як у методі Шелла дуже багато помилок й інколи можна помітити «… 10.382 10.516 18.113 11.214 …». Якщо точність не важлива, а важливий час, то можна використовувати метод Шелла, коли навпаки – тільки метод вибору. До того ж на малих кількостях елементів цей час не помітно.

**Висновок:** Ми отримали практичні навички в обробці масивів, у сортуванні елементів масивів різними методами та за різними реквізитами; дослідили та вивчили методи пошуку ключових елементів у масивах; здійснили порівняння та аналіз ефективності використовуваних методів сортування та пошуку.