**Лабораторная работа № 13. Анализ алгоритмов сортировок**

|  |  |
| --- | --- |
| 1. Написать программу, реализующую алгоритм сортировки *пузырьком* с использованием функции, приведенной в правой части.  Проанализировать алгоритм. | #include <iostream>  using namespace std;  void bubbleSort(int a[], int n)  {  int i, j, t;  for (i = 1; i < n; i++)  for (j = n - 1; j >= i; j--)  if (a[j - 1] > a[j])  {  t = a[j - 1];  a[j - 1] = a[j];  a[j] = t;  }  }  void main()  {  setlocale(LC\_ALL, "Rus");  int size, i, A[100];  cout << "Количество элементов = ";  cin >> size;  for (i = 0; i < size; i++)  {  cout << i + 1 << " элемент = ";  cin >> A[i];  }  bubbleSort(A, size);  cout << "Результирующий массив: ";  for (i = 0; i < size; i++)  cout << A[i] << " ";  cout << endl;  } |
| 2. Разработать главную функцию, реализующую алгоритм *быстрой сортировки Хоара* с использованием функций, приведенных в правой части. | #include <iostream>  using namespace std;  int getHoarBorder(int A[], int sm, int em) {  int i = sm - 1, j = em + 1;  int brd = A[sm], buf;  while (i < j) {  while (A[--j] > brd);  while (A[++i] < brd);  if (i < j) {  buf = A[j];  A[j] = A[i];  A[i] = buf;  }  }  return j;  }  int\* sortHoar(int A[], int sm, int em) {  if (sm < em) {  int hb = getHoarBorder(A, sm, em);  sortHoar(A, sm, hb);  sortHoar(A, hb + 1, em);  }  return A;  }  int main() {  setlocale(LC\_ALL, "ru");  int A[] = { 34, 7, 23, 32, 5, 62, 12, 21 };  int size = sizeof(A) / sizeof(A[0]);  cout << "Исходный массив: ";  for (int i = 0; i < size; i++) {  cout << A[i] << " ";  }  sortHoar(A, 0, size - 1);  cout << "\nОтсортированный массив: ";  for (int i = 0; i < size; i++) {  cout << A[i] << " ";  }  return 0;  } |
| 3. Написать программу, реализующую алгоритм *пирамидальной сортировки* с использованием функций, приведенных в правой части.  Проанализировать алгоритм. | #include <iostream>  using namespace std;  void heapify(int A[], int pos, int n) {  int t, tm;  while (2 \* pos + 1 < n) {  t = 2 \* pos + 1;  if (2 \* pos + 2 < n && A[2 \* pos + 2] >= A[t])  t = 2 \* pos + 2;  if (A[pos] < A[t]) {  tm = A[pos];  A[pos] = A[t];  A[t] = tm;  pos = t;  }  else break;  }  }  void piramSort(int A[], int n) {  for (int i = n - 1; i >= 0; i--)  heapify(A, i, n);  while (n > 0) {  int tm = A[0];  A[0] = A[n - 1];  A[n - 1] = tm;  n--;  heapify(A, 0, n);  }  }  int main() {  setlocale(LC\_ALL, "ru");  int A[] = { 34, 7, 23, 32, 5, 62, 12, 21 };  int size = sizeof(A) / sizeof(A[0]);  cout << "Исходный массив: ";  for (int i = 0; i < size; i++)  cout << A[i] << " ";  piramSort(A, size);  cout << "\nОтсортированный массив: ";  for (int i = 0; i < size; i++)  cout << A[i] << " ";  return 0;  } |
| 4. Написать программу, реализующую алгоритм сортировки *слиянием* с использованием функций, приведенных в правой части.  Проанализировать алгоритм. | #include <iostream>  using namespace std;  void insOrd(int m[], int sm, int em, int e) {  int buf, i = sm;  while (i <= em && m[i] < e) {  if (i - 1 >= sm)  m[i - 1] = m[i];  i++;  }  m[i - 1] = e;  }  int\* merge(int m[], int sm, int cm, int em) {  for (int i = sm; i <= cm; i++) {  if (m[i] > m[cm + 1]) {  int buf = m[i];  m[i] = m[cm + 1];  insOrd(m, cm + 1, em, buf);  }  }  return m;  }  int\* sortMerge(int m[], int lm, int sm = 0) {  if (lm > 1) {  sortMerge(m, lm / 2, sm);  sortMerge(m, lm - lm / 2, sm + lm / 2);  merge(m, sm, sm + lm / 2 - 1, sm + lm - 1);  }  return m;  }  int main() {  setlocale(LC\_ALL, "ru");  int m[] = { 34, 7, 23, 32, 5, 62, 12, 21 };  int size = sizeof(m) / sizeof(m[0]);  cout << "Исходный массив: ";  for (int i = 0; i < size; i++)  cout << m[i] << " ";  sortMerge(m, size);  cout << "\nОтсортированный массив: ";  for (int i = 0; i < size; i++)  cout << m[i] << " ";  return 0;  } |
| 5. В правой части приведена программа, в которой определяется количество временных тактов, требуемых на каждую из некоторых двух сортировок. | #include <ctime>  #include <stdlib.h>  #include <iostream>  using namespace std;  // пузырьком  int\* sort1(int m[], int lm) {  for (int i = 0; i < lm - 1; i++) {  for (int j = 0; j < lm - i - 1; j++) {  if (m[j] > m[j + 1]) {  swap(m[j], m[j + 1]);  }  }  }  return m;  }  // вставками  int\* sort2(int m[], int lm) {  for (int i = 1; i < lm; i++) {  int key = m[i];  int j = i - 1;  while (j >= 0 && m[j] > key) {  m[j + 1] = m[j];  j--;  }  m[j + 1] = key;  }  return m;  }  int getRandKey(int reg = 0) {  if (reg > 0)  srand((unsigned)time(NULL));  int rmin = 0, rmax = 100000;  return (int)(((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin);  }  int main() {  setlocale(LC\_CTYPE, "Russian");  const int N = 50000;  int k[N], f[N];  clock\_t t1, t2;  getRandKey(1);  for (int i = 0; i < N; i++)  f[i] = getRandKey(0);  for (int n = 10000; n <= N; n += 10000) {  cout << "n = " << n << endl;  cout << "Сортировка пузырьком ";  memcpy(k, f, n \* sizeof(int));  t1 = clock();  sort1(k, n);  t2 = clock();  cout << "Прошло " << t2 - t1 << " тактов времени" << endl;  cout << "Сортировка вставками ";  memcpy(k, f, n \* sizeof(int));  t1 = clock();  sort2(k, n);  t2 = clock();  cout << "Прошло " << t2 - t1 << " тактов времени" << endl << endl;  }  return 0;  } |
| В соответствии со своим вариантом написать программу для ***сортировок*** массивов указанными в таблице методами. Исходные массивы заполняются случайными числами.  Определить зависимость времени выполнения алгоритмов от количества элементов для каждого из алгоритмов. Выполнить моделирование для массивов размером 1000, 2000, 3000, 4000, 5000 (в зависимости от быстродействия компьютера размеры массивов можно увеличивать).  Произвести сравнение эффективности алгоритмов (построить графики в приложении Excel). | |
| Ввести массивы А и В. В массив С перенести те элементы массива А, которые больше максимального элемента массива В. Массив С отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», пирамидальная сортировка. | #include <iostream>  #include <ctime>  #include <algorithm>  using namespace std;  void bubbleSort(int a[], int n)  {  int i, j, t;  for (i = 1; i < n; i++)  for (j = n - 1; j >= i; j--)  if (a[j - 1] > a[j])  {  t = a[j - 1];  a[j - 1] = a[j];  a[j] = t;  }  }  void heapify(int A[], int pos, int n)  {  int t, tm;  while (2 \* pos + 1 < n)  {  t = 2 \* pos + 1;  if (2 \* pos + 2 < n && A[2 \* pos + 2] >= A[t])  t = 2 \* pos + 2;  if (A[pos] < A[t])  {  tm = A[pos];  A[pos] = A[t];  A[t] = tm;  pos = t;  }  else break;  }  }  void piramSort(int A[], int n)  {  for (int i = n - 1; i >= 0; i--)  heapify(A, i, n);  while (n > 0)  {  int tm = A[0];  A[0] = A[n - 1];  A[n - 1] = tm;  n--;  heapify(A, 0, n);  }  }  void test(int n) {  int\* a = new int[n];  int\* b = new int[n];  int bmax = INTMAX\_MIN;  srand(time(0));  for (int i = 0; i < n; i++) {  a[i] = pow(rand(),3);  b[i] = pow(rand(),3);  if (bmax < b[i]) {  bmax = b[i];  }  }  int\* c = new int[n];  int csize = 0;  for (int i = 0; i < n; i++) {  if (a[i] > bmax) {  c[csize] = a[i];  csize++;  }  }  int\* temp = new int[n];  for (int i = 0; i < csize; i++) {  temp[i] = c[i];  }  double start = clock();  bubbleSort(c, csize);  double end = clock();  double sec = (end - start) / CLOCKS\_PER\_SEC;  cout << "Время сортировки пузырьком: " << sec << " секунд\n";  start = clock();  piramSort(temp, csize);  end = clock();  sec = (end - start) / CLOCKS\_PER\_SEC;  cout << "Время пирамидальноя сортировки: " << sec << " секунд\n";  cout << '\n';  }  int main() {  setlocale(LC\_ALL, "rus");  test(10000);  test(20000);  test(30000);  test(40000);  return 0;  } |

**Дополнительные задания**

|  |  |
| --- | --- |
| 2. Разработать функцию сортировки **выбором** и добавить к списку исследуемых сортировок. Алгоритм: осуществляется поиск элемента, имеющего наименьшее значение. После того, как этот элемент найден, он меняется местами с первым элементом. Затем, начиная со второго элемента массива, осуществляется поиск следующего наименьшего значения элемента. Найденный элемент меняется местами со вторым элементом. Этот процесс продолжается до тех пор, пока все числа не будут расположены в порядке возрастания.  3. Разработать функцию сортировки методом **простой вставки** и добавить к списку исследуемых сортировок. Алгоритм: массив в процессе сортировки делится на две части: упорядоченную и неупорядоченную. Вначале весь массив неупорядочен. На каждом шаге из неупорядоченной части извлекается первый элемент, который вставляется на нужное место упорядоченной части. При этом размер упорядоченной части увеличивается на единицу. В конце весь массив окажется упорядоченным.  4. Разработать функцию сортировки методом **Шелла** и добавить к списку исследуемых сортировок. Алгоритм: метод Шелла **(**сортировка вставками с убывающим шагом) состоит в том, что упорядочиваемый массив делится на группы элементов, каждая из которых упорядочивается методом простой вставки (см. дополнительное задание 3). В процессе сортировки размеры таких групп увеличиваются до тех пор, пока все элементы не войдут в упорядоченную группу. | #include <iostream>  #include <ctime>  using namespace std;  void selectionSort(int a[], int n)  {  int i, j, minIndex, temp;  for (i = 0; i < n - 1; i++)  {  minIndex = i;  for (j = i + 1; j < n; j++)  if (a[j] < a[minIndex])  minIndex = j;  temp = a[i];  a[i] = a[minIndex];  a[minIndex] = temp;  }  }  void insertionSort(int a[], int n)  {  int i, j, temp;  for (i = 1; i < n; i++)  {  temp = a[i];  j = i - 1;  while (j >= 0 && a[j] > temp)  {  a[j + 1] = a[j];  j--;  }  a[j + 1] = temp;  }  }  void shellSort(int a[], int n)  {  int gap, i, j, temp;  for (gap = n / 2; gap > 0; gap /= 2)  {  for (i = gap; i < n; i++)  {  temp = a[i];  j = i;  while (j >= gap && a[j - gap] > temp)  {  a[j] = a[j - gap];  j -= gap;  }  a[j] = temp;  }  }  }  int main() {  setlocale(LC\_ALL, "rus");  int n = 100000;  int\* t1 = new int[n];  int\* t2 = new int[n];  int\* t3 = new int[n];  srand(time(0));  for (int i = 0; i < n; i++) {  int temp = rand();  t1[i] = temp;  t2[i] = temp;  t3[i] = temp;  }  double start = clock();  selectionSort(t1, n);  double end = clock();  double sec = (end - start) / CLOCKS\_PER\_SEC;  cout << "Время сортировки выбором: " << sec << " секунд\n";  start = clock();  insertionSort(t2, n);  end = clock();  sec = (end - start) / CLOCKS\_PER\_SEC;  cout << "Время сортировки вставкой: " << sec << " секунд\n";  start = clock();  shellSort(t2, n);  end = clock();  sec = (end - start) / CLOCKS\_PER\_SEC;  cout << "Время сортировки Шелла: " << sec << " секунд\n";  return 0;  } |