Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационных технологий и анализа данных

Отчет  
о лабораторной работе № 7

«Классические алгоритмы»

по дисциплине

**«Программирование»**

|  |  |
| --- | --- |
| Выполнил: студент(-ка) гр. | Распутин Г.М. |
| Руководитель: | Лебедева Е.Н. |
| Оценка: |  |
| Дата защиты: |  |

**Иркутск 2024**

### Задание 1 реализовать линейный поиск

1. Условие задачи.

Напишите программу, которая определяет, сколько раз встречается заданное число *x*в данном массиве.

**Входные данные**

В первой строке задается одно натуральное число *N*, не превосходящее 1000 – размер массива.

Во второй строке вводятся *N* чисел – элементы массива (целые числа, не превосходящие по модулю 1000).

В третьей строке содержится одно целое число *x*, не превосходящее по модулю 1000.

**Выходные данные**

Вывести одно число – сколько раз встречается *x*в данном массиве.

1. Пояснения решения

Этот алгоритм перебирает все элементы в массиве, сравнивая их с заданным ключом, из-за полного перебора скорость поиска намного меньше, чем в других алгоритмах.

Я использовал стандартный алгоритм библиотеки C++ **count\_if**,который проверяет соответствие x каждому элемент массива с помощью лямбда-функции и выводит количество соответствий.

1. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Определить сколько раз встречается заданное число x в данном массиве | N=5  mas= {1;2;3;4;5}  X=3 | 1 |
| 2 | Определить сколько раз встречается заданное число x в данном массиве | N=5  mas= {3;3;3;3;3}  X=3 | 5 |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

#include <vector>

using namespace std;

int main() {

int N, a, x;

vector<int> mas;

cin >> N;

for(int i = 0; i < N; i++){

a = 0;

cin >> a;

mas.push\_back(a);

}

cin >> x;

int count = count\_if(mas.begin(), mas.end(), [x](int n) { return n == x; });

cout << count;

return 0;

}

### Задание 2 реализовать бинарный поиск

1. Условие задачи.

Двоичный поиск.

Реализуйте алгоритм бинарного поиска.

**Входные данные**

В первой строке входных данных содержатся натуральные числа N и K (0 < N, K < 100000). Во второй строке задаются N элементов первого массива, отсортированного по возрастанию, а в третьей строке – K элементов второго массива. Элементы обоих массивов - целые числа, каждое из которых по модулю не превосходит 109

**Выходные данные**

Требуется для каждого из K чисел вывести в отдельную строку "YES", если это число встречается в первом массиве, и "NO" в противном случае.

1. Пояснения решения

На каждой итерации вычисляется срединный индекс mid.

Если значение в позиции mid больше a, то обновляется правая граница r на mid, иначе обновляется левая граница l на mid + 1.

Таким образом, каждая итерация сужает диапазон поиска в два раза.

1. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Вывести YES, если элемент есть в массиве и NO, если нет | N=10  K = 5  masN= {1;2;3;4;5;6;7;8;9;10}  mas2=  { -2;0; 4; 9; 12; } | NO  NO  YES  YES  NO |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

int main() {

int l, r, N, K, a;

cin >> N >> K;

vector<int> vecN;

int mid;

for(int i = 0; i < N; i++){

cin >> a;

vecN.push\_back(a);

}

for(int i = 0; i < K; i++){

l = 0;

r = N;

cin >> a;

while (l < r) {

mid = (l + r) / 2; // считываем срединный индекс отрезка [l,r]

if (vecN[mid] > a) r = mid; // проверяем, какую часть нужно отбросить с поиска

else l = mid + 1;

}

if(r != 0) r--;

if (vecN[r] == a) cout << "YES\n";

else cout << "NO\n";

}

return 0;

}

### Задание 3 реализовать пузырьковую сортировки и подсчитать количество обменов

1. Условие задачи.

Пузырьковая сортировка и количество обменов

Определите, сколько обменов сделает алгоритм пузырьковой сортировки по возрастанию для данного массива.

**Входные данные**

На первой строке дано число *N* (1 ≤ *N* ≤ 1000) – количество элементов в массиве. На второй строке – сам массив. Гарантируется, что все элементы массива различны и не превышают по модулю 109.

**Выходные данные**

Выведите одно число – количество обменов пузырьковой сортировки.

1. Пояснения решения

В пузырьковой сортировке каждый элемент сравнивается со следующим. Если два таких элемента не стоят в нужном порядке, то они меняются между собой местами. В конце каждой итерации наибольший/наименьший элемент ставится в конец списка. При каждой итерации счетчик contff увеличивается, подсчитывая количество обменов.

1. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Определите, сколько обменов сделает алгоритм пузырьковой сортировки по возрастанию для данного массива. | N=5  Mas = {1;2;3;4;5} | Contff = 0 |
| 2 | Определите, сколько обменов сделает алгоритм пузырьковой сортировки по возрастанию для данного массива. | N=7  Mas = {3;4;5;1;2;3;8} | Contff = 8 |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

using namespace std;

int main() {

int N, a, contff = 0;

int \*arr = new int[1000];

int temp;

cin >> N;

for (int i = 0; i < N; i++) {

cin >> a;

arr[i] = a;

}

for (int i = 0; i < N - 1; i++) {

for (int j = 0; j < N - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

contff++;

}

}

}

cout << contff;

return 0;

}

### Задание 4 Реализовать сортировку вставками

1. Условие задачи.

Сортировка вставками

Требуется отсортировать массив по неубыванию методом "вставок".

**Входные данные**

В первой строке вводится одно натуральное число, не превосходящее 1000 – размер массива. Во второй строке задаются *N* чисел – элементы массива (целые числа, не превосходящие по модулю 1000).

**Выходные данные**

Вывести получившийся массив.

1. Пояснения решения

В сортировке вставками начинаем со второго элемента. Проверяем между собой второй элемент с первым и, если надо, меняем местами. Сравниваем следующую пару элементов и проверяем все пары до нее.

Этот метод сортировки эффективен для небольших массивов или уже частично отсортированных массивов, так как он выполняет минимальное количество операций для вставки каждого элемента на своё место.

1. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | отсортировать массив по неубыванию методом "вставок" | N=5  Mas = {5;4;3;2;1} | Mas = {1;2;3;4;5} |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

using namespace std;

int main() {

int a, N;

int\* A = new int[1000];

cin >> N;

for(int i = 0; i<N; i++){

cin >> a;

A[i] = a;

}

for(int i=1;i<N;i++)

for(int j=i;j>0 && A[j-1]>A[j];j--) // пока j>0 и элемент j-1 > j, x-массив int

swap(A[j-1],A[j]);

for(int i = 0; i < N; i++){

cout << A[i] << " ";

}

return 0;

}

### Задание 5 Реализовать быструю сортировку

1. Условие задачи.

Быстрая сортировка

Дано N чисел, требуется выяснить, сколько среди них различных.

**Входные данные**

В первой строке дано число N – количество чисел. (1 <= N <= 100000) Во второй строке даны через пробел N чисел, каждое не превышает 2\*109 по модулю.

**Выходные данные**

Выведите число, равное количеству различных чисел среди данных.

1. Пояснения решения

В основе быстрой сортировки лежит стратегия «разделяй и властвуй». Задача разделяется на более мелкие подзадачи. Подзадачи решаются отдельно, а потом решения объединяют. Точно так же, массив разделяется на подмассивы, которые сортируются и затем сливаются в один.

В первую очередь выбираем опорный элемент. Отметим его синим. Все значения больше опорного элемента ставятся после него, остальные — перед.

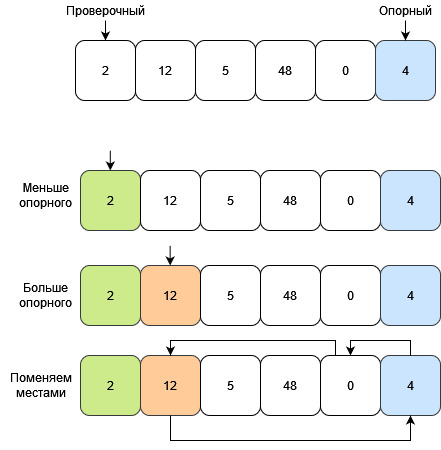
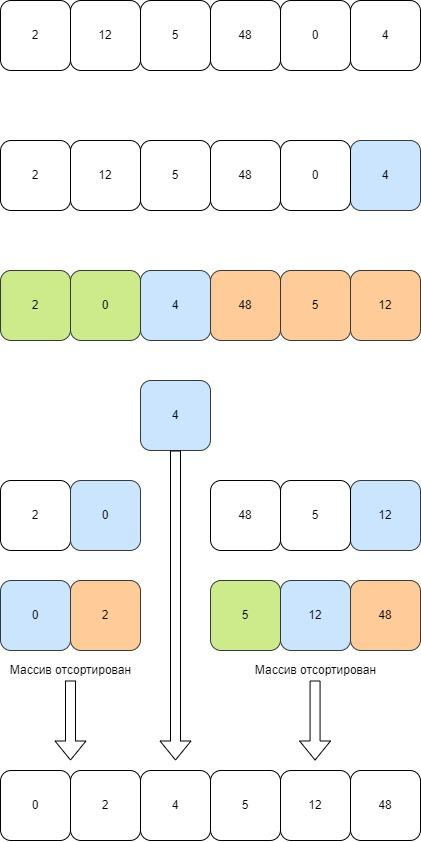
На иллюстрации массив разделяется по опорному элементу. В полученных массивах также выбираем опорный элемент и разделяем по нему.

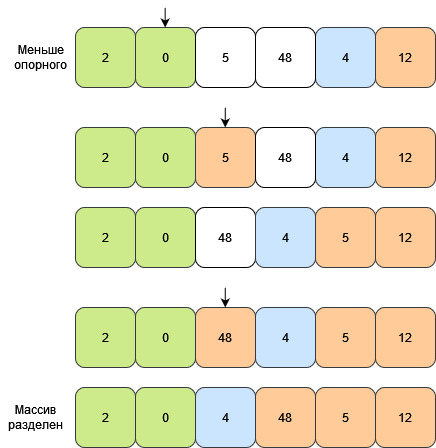
Опорным может быть любой элемент. Мы выбираем последний в списке.

Чтобы расположить элементы большие — справа от опорного элемента, а меньшие — слева, будем двигаться от начала списка. Если число будет больше опорного, то оно ставится на его место, а сам опорный на место перед ним.

функция partition() возвращает индекс опорного элемента, поэтому используем ее в сортировке.

После сортировки проходимся обычным циклом по порядку и сравниваем элементы.





1. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Найти количество различных чисел в массиве | N=5  Mas = {1;0;1;2;0} | 3 |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

using namespace std;

int partition(int list[], int start, int pivot)

{

int i = start;

while(i < pivot)

{

if(list[i] > list[pivot] && i == pivot-1)

{

swap(list[i], list[pivot]);

pivot--;

}

else if(list[i] > list[pivot])

{

swap(list[pivot - 1], list[pivot]);

swap(list[i], list[pivot]);

pivot--;

}

else i++;

}

return pivot;

}

void quickSort(int list[], int start, int end)

{

if(start < end)

{

int pivot = partition(list, start, end);

quickSort(list, start, pivot - 1);

quickSort(list, pivot + 1, end);

}

}

int main()

{

int list[100000];

int N;

cin >> N;

for(int i = 0; i < N; i++){

int a;

cin >> a;

list[i] = a;

}

quickSort(list, 0, N-1);

int conter = 0;

for(int i = 0; i < N-1; i++){

if (list[i] != list[i+1]) conter ++;

}

if (list[N-1] != list[N-2]) conter ++;

std::cout << conter << "\n";

return 0;

}

### Задание 6 реализовать сортировку слиянием

1. Условие задачи.

Сортировка слиянием

Отсортируйте данный массив, используя сортировку слиянием.

**Входные данные**

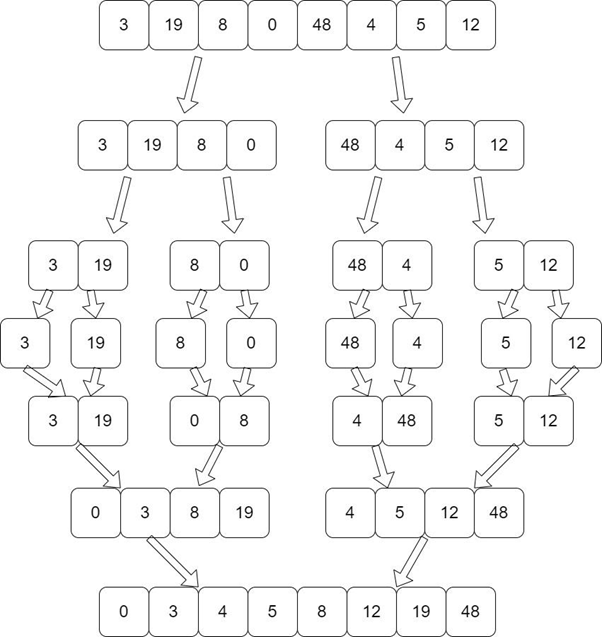
Первая строка входных данных содержит количество элементов в массиве N, N ≤ 105. Далее идет N целых чисел, не превосходящих по абсолютной величине 109.

**Выходные данные**

Выведите эти числа в порядке неубывания.

1. Пояснения решения

Сортировка слиянием также следует стратегии «разделяй и властвуй». Разделяем исходный массив на два равных подмассива. Повторяем сортировку слиянием для этих двух подмассивов и объединяем обратно.



Цикл деления повторяется, пока не останется по одному элементу в массиве. Затем объединяем, пока не образуем полный список.

Алгоритм сортировки состоит из четырех этапов:

1. Найти середину массива.
2. Сортировать массив от начала до середины.
3. Сортировать массив от середины до конца.
4. Объединить массив.
5. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Вывести массив в порядке неубывания | N=5  Mas = {9;4;3;7;1} | 1;3;4;7;9 |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

using namespace std;

void merge(int list[],int start, int end, int mid);

void mergeSort(int list[], int start, int end)

{

int mid;

if (start < end){

mid=(start+end)/2;

mergeSort(list, start, mid);

mergeSort(list, mid+1, end);

merge(list,start,end,mid);

}

}

void merge(int list[],int start, int end, int mid)

{

int mergedList[8];

int i, j, k;

i = start;

k = start;

j = mid + 1;

while (i <= mid && j <= end) {

if (list[i] < list[j]) {

mergedList[k] = list[i];

k++;

i++;

}

else {

mergedList[k] = list[j];

k++;

j++;

}

}

while (i <= mid) {

mergedList[k] = list[i];

k++;

i++;

}

while (j <= end) {

mergedList[k] = list[j];

k++;

j++;

}

for (i = start; i < k; i++) {

list[i] = mergedList[i];

}

}

int main()

{

int N;

cin >> N;

int list[1000000];

for(int i = 0; i < N; i++){

int a;

cin >> a;

list[i] = a;

}

mergeSort(list, 0, N-1);

for (int i = 0; i < N; i++)

{

cout<<list[i]<<" ";

}

}

### Задание 7 Реализовать пирамидальную сортировку

1. Условие задачи.

Сортировка пирамидой

Отсортируйте данный массив. Используйте пирамидальную сортировку.

**Входные данные**

Первая строка входных данных содержит количество элементов в массиве N, N ≤ 105. Далее идет N целых чисел, не превосходящих по абсолютной величине 109.

**Выходные данные**

Выведите эти числа в порядке неубывания.

1. Пояснения решения

Исходный массив представляем в виде структуры данных кучи. Куча – это один из типов бинарного дерева.

У кучи есть следующие свойства:

* Родительский узел всегда больше дочерних;
* На i-ом слое 2i узлов, начиная с нуля. То есть на нулевом слое 1 узел, на первом – 2 узла, на втором – 4, и т. д. Правило для всех слоев, кроме последнего;
* Слои заполняются слева направо.

После формирования кучи будем извлекать самый старший узел и ставить на конец массива.

Алгоритм сортировки кучей:

1. Формируем бинарное дерево из массива.
2. Расставляем узлы в дереве так, чтобы получилась куча (метод heapify()).
3. Верхний элемент помещаем в конец массива.
4. Возвращаемся на шаг 2, пока куча не опустеет.

Обращаться к дочерним узлам можно, зная, что дочерние элементы i-го элемента находятся на позициях 2\*i + 1 (левый узел) и 2\*i + 2 (правый узел).

1. Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Вывести массив в порядке неубывания | N=5  Mas = {7;6;2;3;1} | 1;2;3;6;7 |

1. Кодирование алгоритма или запись алгоритма на языке С++

#include <iostream>

using namespace std;

void heapify(int list[], int listLength, int root)

{

int largest = root;

int l = 2\*root + 1;

int r = 2\*root + 2;

if (l < listLength && list[l] > list[largest])

largest = l;

if (r < listLength && list[r] > list[largest])

largest = r;

if (largest != root)

{

swap(list[root], list[largest]);

heapify(list, listLength, largest);

}

}

void heapSort(int list[], int listLength)

{

for(int i = listLength / 2 - 1; i >= 0; i--)

heapify(list, listLength, i);

for(int i = listLength - 1; i >= 0; i--)

{

swap(list[0], list[i]);

heapify(list, i, 0);

}

}

int main()

{

int N;

cin >> N;

int list[1000000];

for(int i = 0; i < N; i++)

{

int a;

cin >> a;

list[i] = a;

}

heapSort(list, N);

for(int i = 0; i < N; i++)

cout << list[i] << ' ';

}