MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

«KHARKIV POLYTECHNIC INSTITUTE»

Department of Software Engineering and Management Information Technologies

Report from lab № 4

discipline «Algorithm and Data Structures»

Kharkiv

2018

Theme :Basic Data Structures:

**Objective:** to explore algorithms of sorting and binary search.

**Task:**

Develop a program that reads numbers N, M (1 <N, M <256), a sequence of N

keys (integers, real numbers or strings (up to 255 characters) depending on the

variant), and a sequence of M keys. The program saves the first sequence to redblack

trees.

Whenever a new element is added to tree, statistics must be display according

to variant.

1 The minimum element and its color;

2 The maximum element and its color.

After building a tree, results of the following operations must be shown for the

tree and for every element x of the second sequence (according to variant).

1 Does item x exist in the tree and what is its color.

2 Successor(x) and its color.

3 Predecessor(x) and its color.

Usage of ready data structures (e.g., STL) is prohibited, but string

implementations can be used (for example, std::string in C++).

**Progress of the lab:**

Theory : Many algorithms use sorting as an intermediate step. There are many different

sorting algorithms; the choice of a particular situation depends on the length of the

sorted sequence, of degree how much it is already sorted, and the type of available

memory.

Merge sort is based on "divide and conqueror", too. First, we divide the array

into two half smaller parts. Then we sort each half separately. Then we can merge the

15

two ordered array into one. Recursive partitioning of the problem into smaller ones

occurs until as long as the array size is more that 1 (any array of length 1 can be

considered as ordered).

Nontrivial part is to connect the two ordered arrays into one. It is performed by

an auxiliary procedure *Merge* (*A , p ,q , r*) . The parameters of this procedure are

array *A* and numbers *p , q , r* indicating the border of merged parts. The

procedure assumes that *p*<*q*<*r* and that areas *A*[ *p* ..*q*] and *A*[ *q*+1 ..*r* ] are

sorted and merges them into one section *A*[ *p* ..*r* ] .

CODE:

using System;

using System.Text;

namespace Merge\_Sort

{

class List

{

private int[] unsorted = new int[20];

private int[] arr = new int[20];

private int[] sorted = new int[20];

private int cmp\_count; //Number of comparisons

private int mov\_count; //Number of data movements

// Number of elements in array

private int n;

public List()

{

cmp\_count = 0;

mov\_count = 0;

}

void read()

{

// Get the number of elements to store in the array

while (true)

{

Console.Write("Enter the number of elements in the array: ");

string s = Console.ReadLine();

n = Int32.Parse(s);

if (n <= 20)

break;

else

Console.WriteLine("\nArray can have maximum 20 elements.\n");

}

Console.WriteLine("-----------------------");

Console.WriteLine(" Enter array elements ");

Console.WriteLine("-----------------------");

// Get array elements

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

{

Console.Write("<{0}> ", i + 1);

string s1 = Console.ReadLine();

arr[i] = Int32.Parse(s1);

unsorted[i] = Int32.Parse(s1);

}

}

public void m\_sort(int low, int high)

{

int mid;

if (low >= high)

return;

mid = (low + high) / 2;

m\_sort(low, mid);

m\_sort(mid + 1, high);

merge(low, mid, high);

}

public void merge(int low, int mid, int high)

{

int i, j, k;

i = low;

j = mid + 1;

k = low;

while ((i <= mid) && (j <= high))

{

if (arr[i] <= arr[j])

{

sorted[k++] = arr[i++];

}

else

{

sorted[k++] = arr[j++];

}

cmp\_count++;

}

//If there are still some elements in the first sub list

//append them to the new list.

while (i <= mid)

{

sorted[k++] = arr[i++];

mov\_count++;

}

//If there are still some elements in the second sub list

//append them to the new list.

while (j <= high)

{

sorted[k++] = arr[j++];

mov\_count++;

}

//Copy the sorted elements in the original array

for (i = low; i <= high; i++)

{

arr[i] = sorted[i];

mov\_count++;

}

}

int getSize()

{

return (n);

}

void display()

{

Console.WriteLine("-----------------------");

Console.WriteLine(" Sorted array elements ");

Console.WriteLine("-----------------------");

for (int j = 0; j < n; j++)

{

Console.WriteLine(arr[j]);

}

Console.WriteLine("\nNumber of comparisons: " + cmp\_count);

Console.WriteLine("\nNumber of data movements: " + mov\_count);

}

void binary()

{

int n = getSize();

for (int j = 0; j < n; j++)

{

binarySearch(unsorted[j]);

}

}

void binarySearch(int item)

{

// Accept the number to be searched

// Apply binary search

int lowerbound = 0;

int upperbound = n - 1;

// Obtain the index of the middlemost element

int mid = (lowerbound + upperbound) / 2;

int ctr = 1;

while ((item != arr[mid]) && (lowerbound <= upperbound))

/\* Loop to search for the element in the array \*/

{

if (item > arr[mid])

lowerbound = mid + 1;

else

upperbound = mid - 1;

mid = (lowerbound + upperbound) / 2;

ctr++;

}

if (item == arr[mid])

Console.WriteLine("\n" + item.ToString() + " found at position " + (mid + 1).ToString());

else

Console.WriteLine("\n" + item.ToString() + " not found in the array\n");

}

static void Main(string[] args)

{

// Declaring the object of the class

List myList = new List();

// Accept array elements;

myList.read();

// Calling the sorting function

myList.m\_sort(0, myList.getSize() - 1); //First call to merge sort algo.

// Display sorted array

myList.display();

//binary search

myList.binary();

// To exit from the console

Console.WriteLine("\n\nPress Return to exit.");

Console.Read();

}

}

}

:

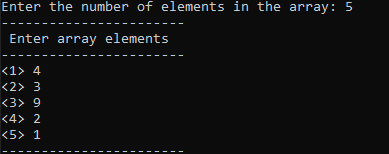


Figure.1- enter size of array and elements of array

\

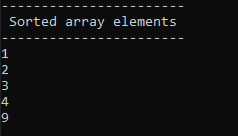


Figure.2- Sorted array

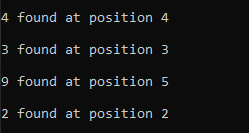


Figure.3- binary search

Conclusion:

In this laboratory the study of merge sorting was considered

* Merge sorting an array.
* Perform binary searching.