YILDIZ TECHNICAL UNIVERSITY

ELECTRICAL- ELECTRONICS FACULTY

COMPUTER ENGINEERING DEPARTMENT

Algorithm Analysis Lecture

First Assignment

KAMRAN BALAYEV 17011904

1. Find closest elements in array with size of N

A) In brute force section application have ***n^2*** complexity

B) In second section i made a merge sort on array and then have used one for loop for searching closest elements in array in this way algorithm complexity was reached to ***Nlog(n)+n*** and as a result ***Nlog(n)+n < n^2***

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
//Prototypes of functions
void bruteForce(int *, int);

int main()
{
    int *arr; //array pointer
    int n;   // array size
    int i;  //loop variable
    int input,stop;//input variable for menu selection, stop variable for stopping program
    int dif;//for storing difference value of array values after merge sort operation
    int first,second;//closest values variables
    //LOOP UNTILL USER ENTER 0
    do{
        printf("Enter array size: ");//ask array size
        scanf("%d",&n);//scan array size
        arr=(int)malloc(n*sizeof(int));//dynamic memory allocation for array
        /* Assign random numbers to the array*/
        printf("Array values: \n");
        for(i=0; i<n; i++){
            arr[i]=rand() %100;
            printf("%d \n",arr[i]);
        };
        printf("\nPlease select 1 for brute force, select 2 for other method: ");
        scanf("%d",&input);
        if(input==1){
            bruteForce(arr,n);//call brute force function
        }
        else if (input ==2){
            mergeSort(arr,0,n-1,n);//sort array
            dif=arr[1]-arr[0];//get difference of 2 values

            for(i=0;i<n;i++){
                //compare first and second difference values
                if(abs(arr[i+1]-arr[i])<dif){
                    dif=abs(arr[i+1]-arr[i]);//assign difference value
                    first=i;//assign first index
                    second=i+1;//assign second index
                }
            }
            printf("\n Difference: %d \n First element %d \n Second element %d\n",dif, arr[first], arr[second]);
        }
        else{
            printf("Please select proper choice!");
        }
        printf("Enter 0 in order to stop program otherwise enter 1 in order to continue: ");
        scanf("%d",&stop);
    }while(stop!=0);

    return 0;
}

void bruteForce(int * arr, int n){
    int i,j;            //loop variables
    int dif, flag;     //value for calculating difference of 2 values
    int first,second; //index variables for
    //initialize value for dif
    dif=arr[0]-arr[1];
    //control if difference is negative or not
    if(dif<0)
        dif*=-1;
    //brute force algorithm
    for(i=0;i<n;i++){
        for(j=i+1;j<n;j++){
            flag=arr[i]-arr[j];//calculate second difference
            if(flag<0)//if difference is negative
                flag*=-1;
            //compare first and second difference values
            if(flag<dif){
                dif=flag;//assign difference value
                first=i;//assign first index
                second=j;//assign second index
            }
        }
    }
    //printf("\nsecond value\n %d",second);
    printf("\n Difference: %d \n First element %d \n Second element %d\n",dif, arr[first], arr[second]);
```

```
    system("pause");
}


// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r, int n)
{

    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m,n);
        mergeSort(arr, m + 1, r,n);

        merge(arr, l, m, r);
    }
}

```

```
Enter array size: 6
Array values:
24
78
58
62
64
5

Please select 1 for brute force, select 2 for other method: 2

 Difference: 2
 First element 62
 Second element 64
Enter 0 in order to stop program otherwise enter 1 in order to continue:
```

```
Enter array size: 5
Array values:
41
67
34
0
69

Please select 1 for brute force, select 2 for other method: 1

 Difference: 2
 First element 67
 Second element 69
Press any key to continue . . .
```

2. Von Neumann’s Neighborhood Problem

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    int n;          //matrix size
    int size;      //size for Von Neumann’s Neighborhood
    int **arr;    //array pointer
    int mdl,i,j;//mdl for finding middle coordinate of array , i and j are loop variables
    int counter=0, rowCount=0; //counter for number of one, rowCount for number of ones per row
    printf("Enter matrix size: ");
    scanf("%d",&n);

    size=2*(n+1)+1; //calculate the centered square number
    /*Dynamic memory allocation*/
    arr=(int*)malloc(size*sizeof(int*));
    for(i=0; i < size; i++) {
        arr[i] = (int )malloc(size * sizeof(int));
    }
    //calculate middle number for starting point
    mdl=(int)size/2;
    //start calculating von Neumann Neighborhood
    for(i=0;i<size;i++){
        for(j=0;j<size;j++){
            arr[i][j]=0;//assign zero to array values
            //if von Neumann neighborhood of range is proper
            if(abs(mdl-j)+abs(mdl-i)<= n){
                arr[i][j]=1;
                counter+=1;//counter for total number of one
                rowCount+=1;//counter for number of ones per row
            }
        }
        //print number of ones per row
        printf("\nnumber of 1 in %d. row:  %d\n", i,rowCount);
        rowCount=0;
    }
     printf("\n");
     for(i=0;i<size;i++){
        for(j=0;j<size;j++){
            printf("%d ", arr[i][j]);//print array elements
        }
        printf("\n");
    }

    printf("\nTotal number of ones: %d \n", counter);
    return 0;
}
```

```
Enter matrix size: 4

number of 1 in 0. row:  0

number of 1 in 1. row:  1

number of 1 in 2. row:  3

number of 1 in 3. row:  5

number of 1 in 4. row:  7

number of 1 in 5. row:  9

number of 1 in 6. row:  7

number of 1 in 7. row:  5

number of 1 in 8. row:  3

number of 1 in 9. row:  1

number of 1 in 10. row:  0

0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0

Total number of ones: 41
```

```
Enter matrix size: 0

number of 1 in 0. row:  0

number of 1 in 1. row:  1

number of 1 in 2. row:  0

0 0 0
0 1 0
0 0 0

Total number of ones: 1
```