Sezin Gümüş 150113841

Homework 1 Part 2
CSE 246 Analysis of Algorithms, Spring 2018

1.(a) An array A[0..n-2] contains n-1 integers from 1 to n in increasing order. (Thus one integer in the range is missing.) Design the most efficient algorithm you can to find the missing integer and indicate its time efficiency.

Answer : Since array A is sorted , binary search implementation can be used to find missing number.

```
find_missing(A[0...n-2])
     int middle = floor (length/2)  //length :length of array A
     int left_side = middle – 0   //number of integers that is less than minimum middle index (left part)
     int right_side = length - middle – 1 //  number of integers that is greater than middle index (right part)
     int right_middle = 0  // middle index of the left part
     int left_middle = 0   //middle index of the right part

     //If the left side is greater than right side , the missing number is in the right part
     if (left_side > right_side)
     {
       while ( middle < length )
       {
         right_middle = (length - middle) / 2
         middle =middle + right_middle
       }

     }

    //If the right side is greater than left side , the missing number is in the left part
     else if (left_side <  right_side)
     {
    while (middle >= left_middle)
    {
       left_middle = (middle -length) / 2
       middle = middle - left_middle
    }

   else return -1
return middle  // gives the missing number  .
```

Time efficiency of this algorithm is Θ(logn) .

1.(b) Repeat (a) for an unsorted array (instead of an array in increasing order).

Answer : To find missing number ,first array A is sorted with MergeSort and same algorithm in section (a) is used for finding missing number in this array . Time efficiency will be Θ(nlogn).

Sezin Gümüş 150113841

2. For each of the following problems design a divide-and-conquer algorithm. Code your algorithm in a programming language of your choice.For each of the problem give the following:
i.Step-by-step description of your algorithm.
ii. Code (No need to give in a seperate file, just a printed copy inside the HW report)
iii. At least three sample input/output.

(a) Problem 1: Consider an integer list A[0..n-1] that includes negative and positive integers. Among all subsequences in this list, find the sum of the subsequence that has largest sum.

Answer : For this problem , first a binary tree is built ,a middle element of the array A is set as root of this tree , then rest of the elements will placed in binary tree . After that , right nodes of this tree is sum . If the right tree has only left child , then ,this node will be picked too . To see the code of this algorithm with detail , see the Codes – Inputs and Outputs **,** 2.a part .

(b) Problem 2: Consider an integer list A[0..n-1] that includes negative and positive integers. Find the longest all-negative subsequence

Answer : For this problem , array A is divided into two parts which are left and right . In the each parts , longest path are checked , after that middle element is checked . If the middle element of array A is negative number and next and previous elements are negatives , this means , the longest path begins from left side and ends in the right part . To count max element To see the code of this algorithm with detail , see the Codes – Inputs and Outputs **,** 2.b part .

(c) Problem 3: Consider a binary list B[0..n-1] that includes 0's and1's. Find the length of the longest alternating subsequence 010101......

Answer : Solution of this problem is similar as problem two .The array B is divided into two parts which are left and right . In the each parts , longest alternating path are checked , after that middle element is checked . If the middle element of array A is different from the next and previous element , the longest path may begin from left side and ends in the right part . To see the code of this algorithm with detail , see the Codes – Inputs and Outputs **,** 2.c part .

3. Let A = { a 1 , ... a n } and B = { b 1 , ... b m } be two sets of numbers and $m=n^2$. Consider the problem of finding their intersection, i.e., the set C of all the numbers that are in both A and B. Design an efficient algorithm for solving this problem and determine its efficiency class in terms of n.

Answer : For this problem , first , Array A is sorted with MergeSort algorithm ,then each element of array B is searched in array B by using binary search algorithm. Time efficiency of this algorithm to solve this problem is $\Theta(n^2 \log n)$.

Sezin Gümüş 150113841

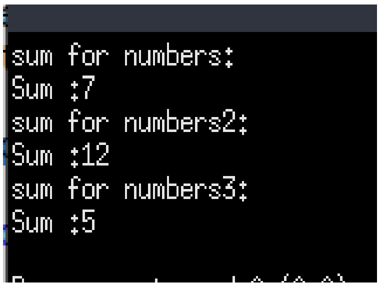
Codes – Inputs and Outputs

2.a

Inputs :

numbers [8] = {-2,-5,6,-2,-3,1,5,-6};
numbers2 [9] = {-2,-5,6,-2,-3,1,5,5,-8};
numbers3 [6] = {-2,-2,-3,1,5,4};

Outputs :



Code :

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

struct treeNode {
struct treeNode *leftPtr; /* pointer to left subtree */
int data; /* node value */
struct treeNode *rightPtr; /* pointer to right subtree */
}; /* end structure treeNode */
typedef struct treeNode TreeNode; /* synonym for struct treeNode */
typedef TreeNode *TreeNodePtr; /* synonym for TreeNode* */
/* prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
void inOrder( TreeNodePtr treePtr );
void preOrder( TreeNodePtr treePtr );
void postOrder( TreeNodePtr treePtr );


int main( void ){
int i; /* counter to loop for construct binary tree */
TreeNodePtr rootPtr = NULL; /* tree initially empty */
TreeNodePtr root2Ptr = NULL; /* tree initially empty */
TreeNodePtr root3Ptr = NULL; /* tree initially empty */

//Middle element of the arrays
int  numbers [8] = {-2,-5,6,-2,-3,1,5,-6};
int length = sizeof(numbers)/sizeof(int);
int pivot = ceil(length/2);
```

```
int  numbers2 [9] = {-2,-5,6,-2,-3,1,5,5,-8};
int length2 = sizeof(numbers2)/sizeof(int);
int pivot2 = ceil(length2/2);

int  numbers3 [6] = {-2,-2,-3,1,5,4};
int length3 = sizeof(numbers3)/sizeof(int);
int pivot3 = ceil(length3/2);

//middle element is put to the root of the binary search tree

insertNode(&rootPtr,numbers[pivot]);
insertNode(&root2Ptr,numbers2[pivot2]);
insertNode(&root3Ptr,numbers3[pivot3]);

//Adding elements of the arrays in a binary Search Tree
for ( i = 0; i < length; i++ ){
if (i != pivot)
insertNode( &rootPtr, numbers[i] );
}

for ( i = 0; i < length2; i++ ){
if (i != pivot2)
insertNode( &root2Ptr, numbers2[i] );
}

for ( i = 0; i < length3; i++ ){
if (i != pivot2)
insertNode( &root3Ptr, numbers3[i] );
}

printf("sum for numbers:\n");
find_max_sum(rootPtr);

printf("sum for numbers2:\n");
find_max_sum(root2Ptr);

printf("sum for numbers3:\n");
find_max_sum(root3Ptr);
return 0;
}

//This method finds the sum of the subsequence that has largest sum .

void find_max_sum (TreeNodePtr treeptr)
{
if (treeptr != NULL)  //The tree is not empty
{
int sum = 0;
while (treeptr != NULL)
```

```
{
 sum += treeptr->data;
  //If the right child of the node is empty but still has a left child , this element will be
  //added to sum.
 if(treeptr->rightPtr == NULL && treeptr->leftPtr != NULL)
 {
  //Elements of
   treeptr = treeptr->leftPtr;
   sum += treeptr→data; }
 treeptr = treeptr→rightPtr; }
 printf("Sum :%d \n",sum); }
else //If the tree is empty
{
  printf("Tree is empty \n");
  return -1; }
}
void insertNode( TreeNodePtr *treePtr, int value ) {
/* if tree is empty */
if ( *treePtr == NULL ) {
*treePtr = malloc( sizeof( TreeNode ) );
/* if memory was allocated then assign data */
if ( *treePtr != NULL ) {
( *treePtr )->data = value;
( *treePtr )->leftPtr = NULL;
( *treePtr )->rightPtr = NULL;
} /* end if */
else {
printf( "%d not inserted. No memory available.\n", value );
} /* end else */
} /* end if */
else { /* tree is not empty */
/* data to insert is less than data in current node */
if ( value <= ( *treePtr )->data ) {
insertNode( &( ( *treePtr )->leftPtr ), value );
} /* end if *
/* data to insert is greater than data in current node */
else  {
insertNode( &( ( *treePtr )->rightPtr ), value );
} /* end else if */
/* end else */
} /* end else */
} /* end function insertNode */
```

2.b

Inputs:
numbers [10] = {2,5,0,3,5,0,10,2,1,2};
numbers1 [10] = {2,5,0,-3,-5,0,-1,-2,-1,2};
numbers2 [10] = {2,5,0,-3,-5,-3,-1,-2,-1,2};

Sezin Gümüş 150113841

Outputs:

```
Length of longest negative path for numbers is :0
Length of longest negative path for numbers1 is :3
Length of longest negative path for numbers2 is :6
```

Code:

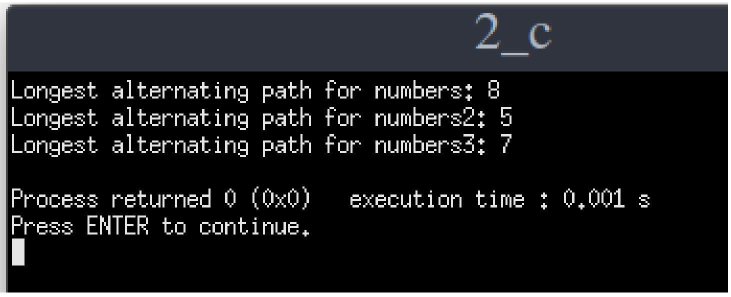
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main( void )
{
//Inputs and outputs
int numbers [10] = {2,5,0,3,5,0,10,2,1,2};
int length = sizeof(numbers)/sizeof(int);
printf("Length of longest negative path for numbers is :%d \n",longest_negative_path(numbers,length));

int numbers1 [10] = {2,5,0,-3,-5,0,-1,-2,-1,2};
int length1 = sizeof(numbers1)/sizeof(int);
printf("Length of longest negative path for numbers1 is :%d
\n",longest_negative_path(numbers1,length1));

int numbers2 [10] = {2,5,0,-3,-5,-3,-1,-2,-1,2};
int length2 = sizeof(numbers2)/sizeof(int);
printf("Length of longest negative path for numbers2 is :%d
\n",longest_negative_path(numbers2,length1));

return 0;
}

/*middle  = middle element of index of the array
max_l = longest negative path of left path
max_r = longest negative path of right path
c_left = counter for negative path in the left side of the array
c_right = counter for negative path in the right side of the array
max_total = longest negative path in this array*/

int longest_negative_path (int number [] , int length)
{
int i;
int middle = ceil(length/2);
int max_l = 0, max_r = 0,c_left=0,c_right=0 , max_total;

//Finds longest negative path in left part , number [0 to middle]

for (i= 0;i<middle;i++)
{
```

```
 if (number[i] < 0 && number [i+1] <0)
 {
  c_left ++;
  if (c_left > max_l) //If there is a corrupted negative path , max_l will keep previous longest negative path
    max_l = c_left;
 }
 else
  {
   c_left = 0;   //If a positive number comes next element , counter will be set to 0.
  }
}

//Finds longest negative path in right part , number [middle to length]

for (i= middle+1;i<length;i++)
{
 if (number[i] < 0 && number [i+1] <0)
 {
  c_right ++;
  if (c_right > max_r)
   max_r = c_right; //If there is a corrupted negative path , max_l will keep previous longest negative path
 }

 else
  {
   c_right = 0;  //If a positive number comes next element , counter will be set to 0.
  }
}

/*Middle element of the array is checked , if it is a negative number
next and previous of the elements will be checked to see whether
they are negative or not .If the condition is provided , then negative path
begins from left part and ends in right path*/

/*If the longest path is in the left part , then max_l will be longest path*/
/*If the longest path is in the right part , then max_r will be longest path*/
if (number[middle] < 0 && number[middle-1] < 0 && number[middle+1] <0)
 max_total = max_l + max_r + 2;
else
{
 if (max_l > max_r)
  max_total = max_l + 1;
 else if (max_l == 0 && max_r == 0)
  max_total = 0;
 else
  max_total = max_r + 1 ;
}

return max_total;
}
```

2.c

Inputs:
int numbers[15] = {0, 1, 0, 1, 0, 0, 0, 1, 0,1,0,1,0,1,1};
int numbers2 [8] = {1, 1, 1, 0,1 ,0 , 1, 1};
int numbers3 [15] = {0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1};

Outputs:



Code :

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

//2. question part c

int main( void )
{

//Inputs and outputs
int numbers[15] = {0, 1, 0, 1, 0, 0, 0, 1, 0,1,0,1,0,1,1};
int length = sizeof(numbers)/sizeof(int);
printf("Longest alternating path for numbers: %d\n",longest_alternating_path(numbers,length));

int numbers2 [8] = {1, 1, 1, 0,1 ,0 , 1, 1};
int length2 = sizeof(numbers2)/sizeof(int);
printf("Longest alternating path for numbers2: %d\n",longest_alternating_path(numbers2,length2));

int numbers3 [15] = {0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1};
int length3 = sizeof(numbers3)/sizeof(int);
printf("Longest alternating path for numbers3: %d\n",longest_alternating_path(numbers3,length3));
return 0;
}
//Returns longest alternating path
int longest_alternating_path (int number [] , int length)
{
int i; //For loop
```

Sezin Gümüş 150113841

```
/*middle  = middle element of index of the array
max_l = longest alternating path of left path
max_r = longest alternating path of right path
c_left = counter for alternating path in the left side of the array
c_right = counter for alternating path in the right side of the array
max_total = longest alternating path in this array*/



int middle = ceil(length/2);
int max_l = 0, max_r = 0,c_left=0,c_right=0 , max_total;

//Finds longest alternating path in left part , number [0 to middle]

for (i= 0;i<middle;i++)
{
 if (number[i] != number [i+1] )
 {
  c_left ++;
  if (c_left > max_l)
    max_l = c_left;  /*If there is a corrupted alternating path , max_l will keep previous longest
                              alternating path*/
 }
 else
  {
   c_left = 0; //If the current element is equal to the next element , counter will be set to 0.
  }
}

//Finds longest alternating path in right part , number [middle to length]

for (i= middle+1;i<length-2;i++)
{
 if (number[i] != number [i+1] )
 {
  c_right ++;
  if (c_right > max_r)
   max_r = c_right;  //If there is a corrupted alternating path , max_r will keep previous longest alternating
path
 }

 else
  {
   c_right = 0; //If the current element is equal to the next element , counter will be set to 0.
  }
}

/*Middle element of the array is checked , if this element is not equal to the previous and next element ,
the longest alternating path may begin from left and end with right part . To see this  , if counter is not
zero ,
```

we can say that this path begins from left and end with right part . */

```
/*If the longest path is in the left part , then max_l will be longest path*/
/*If the longest path is in the right part , then max_r will be longest path*/

if ( number[middle] != number[middle+1] && number[middle] != number [middle-1] )
 max_total = c_left + max_r + 2;


else
{
 if (max_l > max_r)
  max_total = max_l + 1;
 else if (max_l == 0 && max_r == 0)
  max_total = 0;
 else
  max_total = max_r + 1 ;
}
return max_total;
}
```