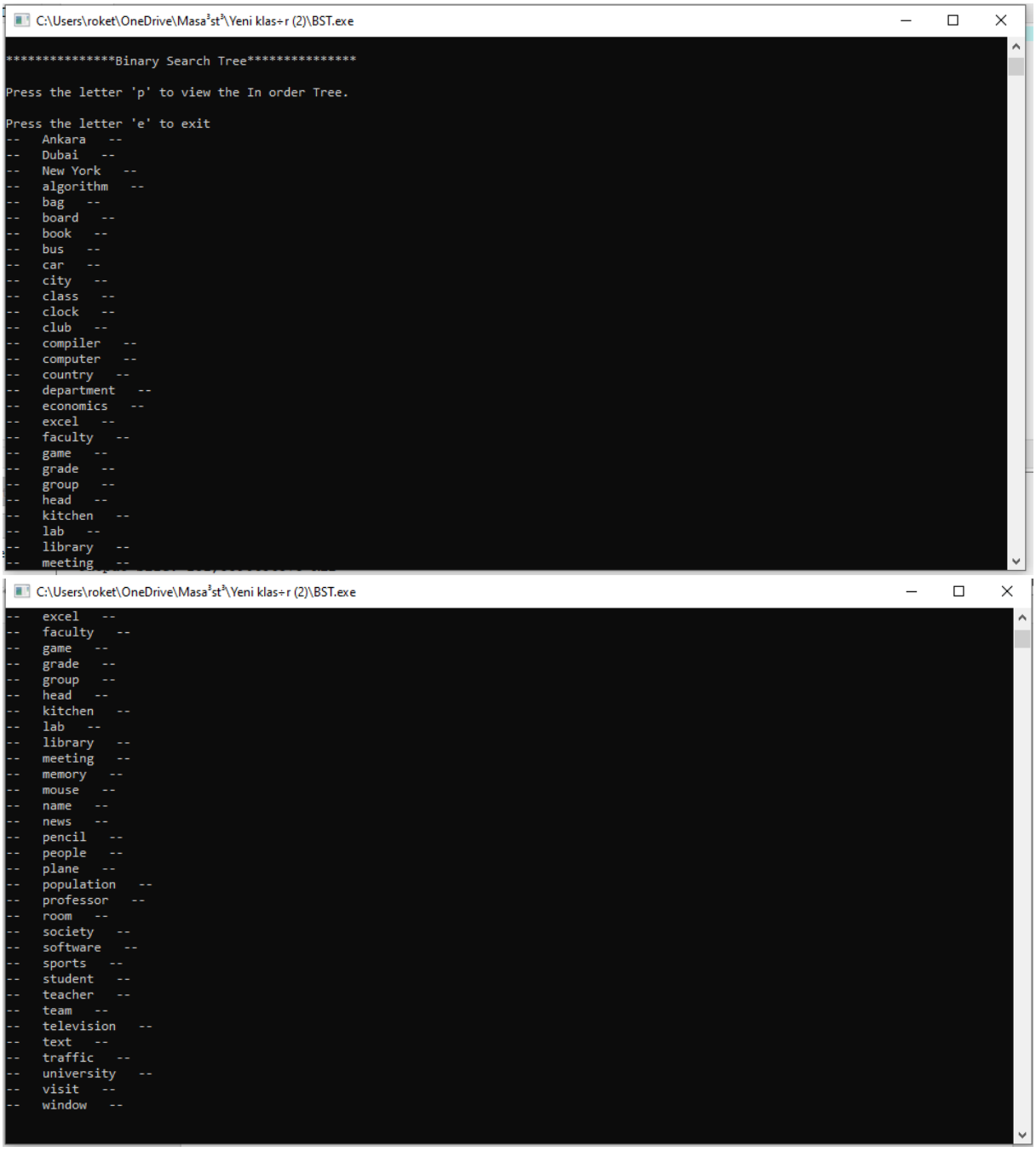
PROJECT REPORT

In the project we were asked to build a 'Binary Search Tree'. The required values for 'Binary Search Tree' are given in advance. I first started the project by setting up a 'Binary Search Tree' that can be built on character. Then I put these values into 'Binary Search Tree' after I made the code to pull the values into the program. I coded the in order version of the tree I made and the outuput was as follows:



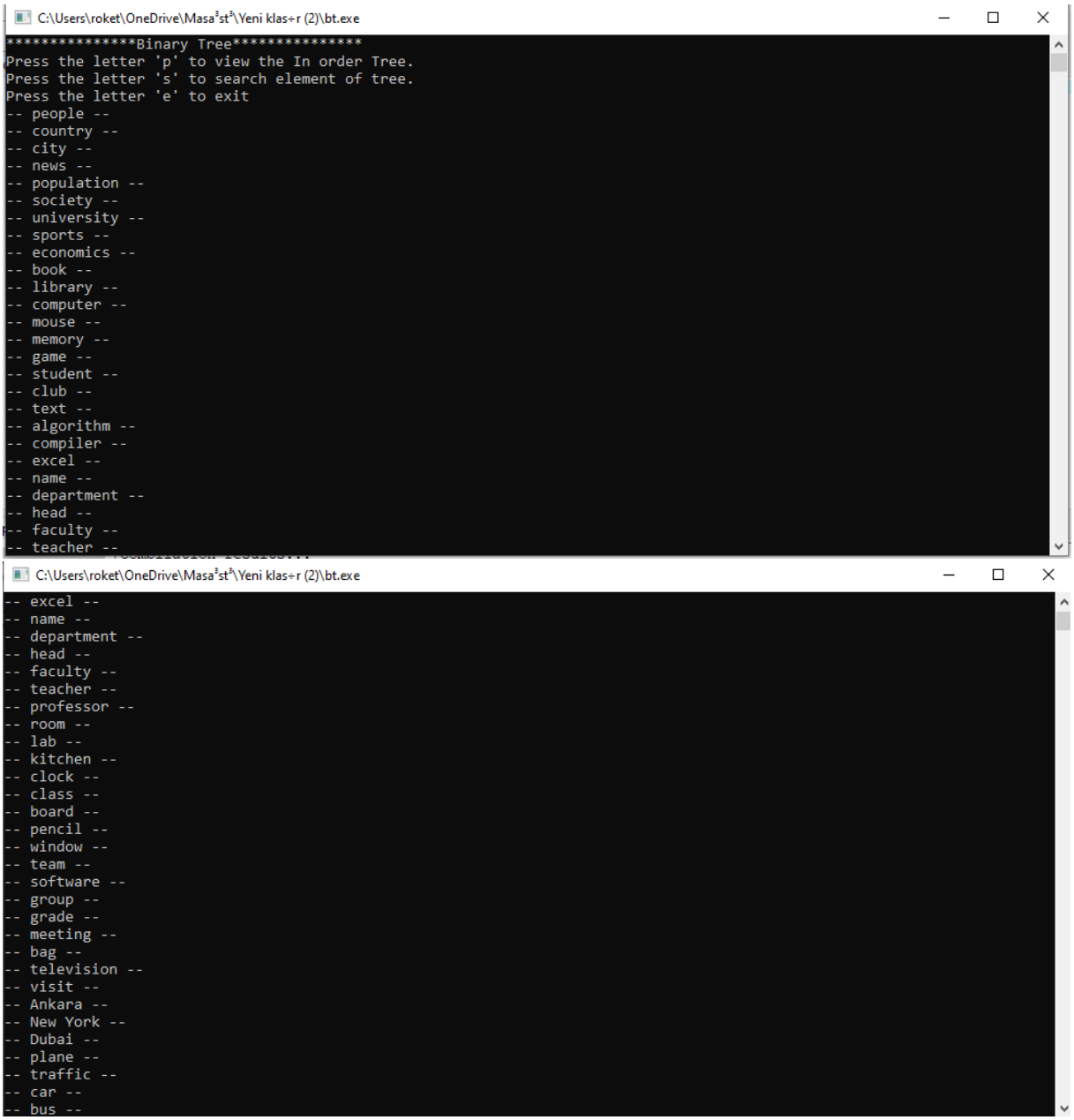
After that I found, I wrote the code to calculate 'Total Access Time', but since I could not get the values properly I typed it manually and found 'Total Access Time''. Its output is as follows:

```
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <=j; k++)
    s += freq[k];
    return s;
}

int main()
{
    int keys[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,3
    int freq[] = {6,10,15,2,1,28,35,62,4,89,3,7,16,27,50,60,70,83,46,44,49,51,56,54,22,33,100,201,92
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ",
                optimalSearchTree(keys, freq, n));
    return 0;
}
```

input

```
Cost of Optimal BST is 12512

...Program finished with exit code 0
Press ENTER to exit console.
```

After my calculation is over. I started to building 'Binary Tree'. I inject input data to 'Binary Tree' and got the output. The output is as follows;



After that I tried to do my search command but I guess it doesn't work correctly.

I will explain my all work end of the page.

I couldn’t do last of part d), that’s why I can’t discuss my results. However, Binary search tree is better than binary tree because binatry search tree has sorted. I mean, the program works more efficiently and more functionally.

```
#include <stdio.h>
#include <stdlib.h>          →          (to use of code)
#include <string.h>

struct node {   → struct for node
   char *value;      →      every void* types changed by char*
   struct node *p_left; → left of the tree
   struct node *p_right; → right of the tree
};
typedef int (*Check)(const char *, const char *); → using typedef to calling the Check function simpler.

void insertingdata(char* key, struct node** subRoot, Check match) →inserting datas elements into the binary search tree
{
   int res; → define a variable
   if( *subRoot == NULL ) {  → to create sub root
      *subRoot = (struct node*) malloc( sizeof( struct node ) ); → memory for struct node
      (*subRoot)->value = malloc( strlen (key) +1 );    →  memory for key
      strcpy ((*subRoot)->value, key);                  → copy the key
      (*subRoot)->p_left = NULL;  → insert and to put left
      (*subRoot)->p_right = NULL;  → insert and to put right
       printf(  "\nnew node for %s" , key); → showing which node inserting
   } else {
      res = match (key, (*subRoot)->value); → to matching (compare) 2 insert element
      if( res < 0)
```

insertingdata(key, &(*subRoot)->p_left, match); → end of the matching if inserting data is small, inserting data will put the tree of left

else

insertingdata(key, &(*subRoot)->p_right, match); → end of the matching if inserting data is big, inserting data will put the tree of left

}

}

int matchString(const char *value1, const char *value2)→ Checks value of the new node against the previous node

{

return (strcmp (value1, value2)); → string comparison instead of pointer comparison

}

void inorder(struct node *Root) → recursive function to print out the tree inorder

{

if(Root != NULL) {

inorder(Root->p_left);

printf("-- %s --\n", Root->value); → string type

inorder(Root->p_right);

}

}

void menu()→displays menu for user

{

printf("**********Binary Search Tree**********\n");

printf("Press the letter 'p' to view the In order Tree.\n");

printf("Press the letter 'e' to exit\n");

}

int sum(int freq[], int i, int j); → A utility function to get sum of array elements

```
int optCost(int freq[], int i, int j)
{
  if (j < i)
    return 0;
  if (j == i)
    return freq[i];
  int fsum = sum(freq, i, j); → Get sum of freq[i], freq[i+1], ... freq[j]
  int min = INT_MAX; →  Initialize minimum value
  int r;
  for (r = i; r <= j; ++r)
  {
      int cost = optCost(freq, i, r-1) +
                 optCost(freq, r+1, j);
      if (cost < min)
        min = cost;
  }
  return min + fsum;  →  Return minimum value
}
int optimalSearchTree(int keys[], int freq[], int n)
{
            return optCost(freq, 0, n-1);
}
int sum(int freq[], int i, int j)
{
   int s = 0;
   int k;
   for (k = i; k <=j; k++)
      s += freq[k];
   return s;
}
int main()
```

```
{
   struct node *p_Root = NULL; → building node
   char *value;
   char option = 'a'; → default option
   FILE* fp; → to take values
   char  line[255];
   fp = fopen("input.txt" , "r"); → to open and read the file
   while (fgets(line, sizeof(line), fp) != NULL)
   {
      char val1[20] ,key[20];
      char *pos;
      if ((pos=strchr(line, '\n')) != NULL)
         *pos = '\0';
      strcpy(val1 , strtok(line,","));
      strcpy(key , strtok(NULL,","));
      insertingdata(key,  &p_Root, (Check)matchString);
   }
   while( option != 'e' ) {
      menu();
      option = getch(); → to input
      if( option == 'p' ) {
         inorder(p_Root);
      }
      else if( option == 'c' )
                  {
                  printf("\nStarted to calculate Cost of Optimal BST.\n");
      int keys[] =
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,
37,38,39,40,41,42,43,44,45,46,47,48,49,50};
                  int freq[] =
{6,10,15,2,1,28,35,62,4,89,3,7,16,27,50,60,70,83,46,44,49,51,56,54,22,33,100,201,92,94,97,93,64,65,
61,19,13,14,26,88,99,205,300,74,77,76,41,42,43,75};
```

```
            int n = sizeof(keys)/sizeof(keys[0]);
            printf("Cost of Optimal BST is %d ",
             optimalSearchTree(keys, freq, n));
        }

        else if( option == 'e' ) {
           printf("Quitted..\nGood Bye ;)");
           exit(0); → to finish the program
        }
    }
return 0;
```