Hacettepe University
Department of Computer Engineering

BBM 201 – Data Structures
Three-Course Exam - October 6, 2020

Problem	Points	Grade
1	15	
2	20	
3	25	
4	20	
5	20	
Total	100	

HONOR STATEMENT

I pledge on my honor that I will not give nor receive any unauthorized help on this exam, and that all submitted work I will upload will be my own.

INSTRUCTIONS

- You have exactly **120 minutes** to finish and submit your answers to the exam.
- This is a regular written exam for which you will turn in your **handwritten** answers using the provided Google Form link.
- This is an open-book exam. You may make use of books, notes, and other online or offline resources. However, **your answers must be your own**. In other words, **your answers MUST NOT be copied from some other resource/answer**. In case your submitted answers are substantially similar to those submitted by other students or those provided by other resources, it will invalidate your exam and trigger a disciplinary investigation.
- You must solve the questions on physical sheets of paper legibly (**easy to read**) with your own **handwriting**. You will upload your answers to the exam submission form by scanning or taking a picture of each answer sheet. Your answers should be complete, understandable and readable in the images you upload. **Typed submissions or unreadable submissions would be regarded as void**.
- Only image files (such as JPG or PNG) can be uploaded to the submission form. When you need to upload your answer in multiple files, you must clearly indicate their order on the answer sheets.
- For each question, you should **only** use the data structures, definitions and functions provided (or explicitly allowed) with that question. You **MUST NOT** use other data structures, definitions or functions, which are not explicitly allowed in the question.

QUESTION 1

(15 points)

(a) **(5 points)** Find what RecursiveFunc (given below) does **in general**.

Finds the maximum of elements in a

(b) **(10 points)** Print the output of the following program.

```
int RecursiveFunc (int array[], int index, int n)
{
     int val1, val2;
     if ( n==1 )
          return array[index];
     val1 = RecursiveFunc (array, index, n/2);
     val2 = RecursiveFunc (array, index+(n/2), n-(n/2));
     if (val1 > val2)
          return val1;
     else
          return val2;
}
...
int a[8] = { 1,2,10,15,16,4,8,2 };
printf( "value is % d \n", RecursiveFunc(a,0,8));
```

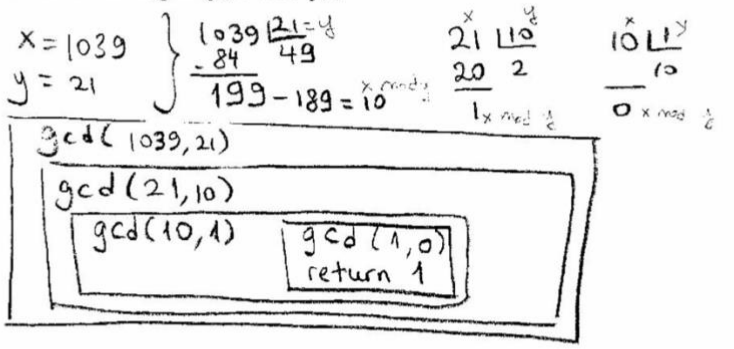
value is 16

QUESTION 2
(20 points)

(a)

(10 points) Below is a pseudocode of the Euclidean algorithm that calculates the greatest common divisor (gcd) of any given two integers. Write all recursive calls made to calculate `gcd(1039, 21)` by using a *box diagram.*

```
algorithm gcd(x,y)
  if y = 0
    then return(x)
  else return(gcd(y,x mod y))
```



(b)

(10 points) Write the complexity of the running time of the following code fragment using big-Oh notation. Show your work.

```
int count = N, i = 0, j = 0;
for (i = 0; i<count; i++)
     if(i%2==0)
          j++;
     else
          j--;
for (i = count; i > 0; i = i/2)
     for (j = 0; j < i; j++)
          count ++;
```

$$\left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{N}{2} \right\rceil + N + \frac{N}{2} + \frac{N}{4} + \cdots + \frac{N}{2^k} =$$

$$k = \log_2 N \qquad = O(N)$$

QUESTION 3
(25 points)

How many array accesses do the following code fragments make as a function of N?

(a) (8 points)

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j++)
        if (a[i] + a[j] == 0)
            count++;
```

$$2\cdot\left[(N-1)+(N-2)+\cdots+1\right]=2\binom{N}{2}=\frac{N(N-1)}{2}\cdot 2$$

(b) (8 points)

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j = 2*j)
        if (a[i] + a[j] == 0)
            count++;
```

1

$$2\left[\overbrace{\overset{i=0}{\log_2(N-1)}+\overset{i=1}{\log_2(N-1)}+\cdots+\overset{i=N-1}{\log_2(N-1)}}^{N\text{ times}}\right]$$

$$\sim 2N\log_2 N$$

(c)

(9 points) What will be the output when doSomething(2) is called?

```
void doSomething(int value)
{
    if(0 < value && value < 10)
    {
        doSomething(value - 1);
        doSomething(value + 1);
        printf(" %d", value);
    }
}
```

$$\begin{tikzcd}
 & & dS(2) \arrow[dl] \arrow[dr] & & \\
 & dS(1) \arrow[d] & & dS(3) \arrow[d] \arrow[dr] & \\
 & dS(2) \arrow[dl] \arrow[dr] & & dS(2) & dS(4) \\
dS(1) \arrow[dr] & & dS(3) \arrow[d] \arrow[dr] & & \\
 & dS(2) & dS(2) & dS(4) &
\end{tikzcd}$$

Nothing is printed.

Never Stops

QUESTION 4
(20 points)

(a) (10 points) A palindromic word is a sequence of characters that reads the same backward and forward. For example; *pepper*, *refer*, *kayak* are palindromic words. The method IsPalindrome(char[] str) checks if a given string is palindrome by using a stack. According to the given prototypes of stack methods, please fill in the gaps in the code.

```
int stack[10];
int top=-1;
void push(char); //pushes the given char on the stack.
char pop();      //pops a char from the stack.

void IsPalindrome(char str[]){
      int len = strlen(str), i, count=0;
      len = strlen(str);

      for (i = 0; i < len; i++)
         push(str[i]);

     for (i = 0; i < len; i++)
         if (str[i] == pop())
             count++;

     if (count==len)
         printf("\n%s is a Palindrome string\n", str);
      else
         printf("\n%s is not a palindrome string\n", str);
}
```

(b) (10 points) Postfix is preferable than infix for computers because of having no checks for parenthesis and no check for operator precedence rules. Prefix has also the same properties as postfix for having no parenthesis and precedence check. Give a reason why postfix is more preferable than prefix for evaluation of expressions by the computers?

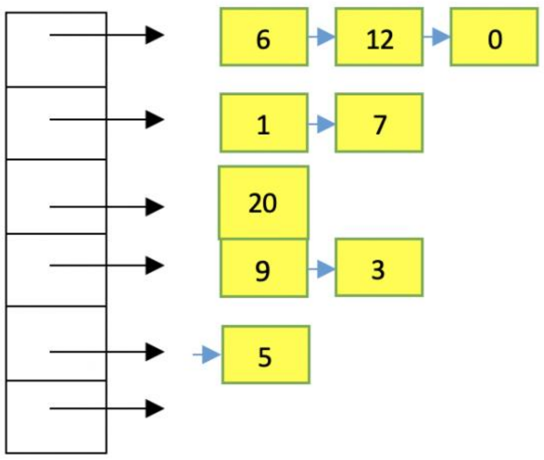
We can evaluate a postfix notation by using only one stack, whereas in prefix notation we have to use at least two stacks for the evaluation. In addition, we need to traverse to the end of the prefix notation and then we need to calculate the expression backwards.

QUESTION 5
(20 points)

```
#define TABLE_SIZE 6

typedef struct list *list_pointer;
typedef struct list {
    int key;
    list_pointer link;
}list;
list_pointer hash_table[TABLE_SIZE];

int hash_function(int key){
      return key % TABLE_SIZE;
}
void insert(int key)
{
      int hash_value = hash_function(key);
      list_pointer ptr, trail=NULL, lead=hash_table[hash_value];
      for (; lead; trail=lead, lead=lead->link)
            if (lead->key==key) {
                  printf("The key is in the table\n");
                  exit(1);
            }
       ptr = (list_pointer) malloc(sizeof(list));

       ptr->key = key;
       ptr->link = NULL;
       if (trail) trail->link = ptr;
       else hash_table[hash_value]= ptr;
}
```

According to the above hashtable implementation, fill in the below figure after the following method calls are applied:

insert(9); insert(20); insert(6);insert(12); insert(3); insert(5); insert(0); insert(1); insert(7);


6
12
0
1
7
20
9
3
5