A

BRAC UNIVERSITY
Department of Computer Science and Engineering

Examination: Mid Semester Exam | Semester: Fall 2023
Duration: 1 Hour 15 Minutes | Full Marks: 40

CSE 221: Algorithms

Answer the following questions.
Figures in the right margin indicate marks.

Name:	ID:	Section:

1. **a. CO2** Explain the time complexity of the following code snippet in regards of the Big-O notation: **04**

```
1.   for (i=0; i<n; i+=4) {
2.       for (j=1; j<n; j*=2) {
3.            for (k=0; k<30; k++) {
4.                 print("Am I still not 30?!!");
5.            }
6.            print("Why, God, why? We had a Deal!");
7.            for (m=n; m>0; m-=2) {
8.                 print("Could you BE more dramatic?");
9.            }
10.      }
11.   }
```

Solution: $n^2 * \log_2 n$

b. CO2 Consider the following functions.

$f_1(n) = (\log n)^{2023}$

$f_2(n) = n^2 \log_n(n^n)$

$f_3(n) = n^3 + 7n^2$

$f_4(n) = 2.023^n$

$f_5(n) = n \log n$

$f_6(n) = n * \sqrt[3]{n^2}$

Now do the followings:

a. Write a correct asymptotic upper bound for each of the above. **03**

03

b. Sort the functions in ascending order of their growth rate, assuming n is significantly large. Just write the sorted order, no need to show any simulation.

Solution: $f_1 < f_5 < f_6 < f_3 = f_2 < f_4$

2. **CO3**

Consider an array containing **N** unique values where for some index **i**, the values are in increasing order from index **0** to (**i-1**), and then again from **i** to **(N-1)**. An example array is given below.
Here **i=3**, it means the values are in increasing order from index 0 to 2, and then again from 3 to 7.

index	0	1	2	3	4	5	6	7
value	9	12	15	2	4	5	7	8

Given the value of **i**, propose an algorithm to search a *key_value* in the array. Complexity of your algorithm must be less than **O(N)**.

a) Present your solution idea as a code/ pseudocode/ flowchart/ step-by-step instructions/ logical explanations in short. **04**
b) Write the time complexity of your presented solution. **01**
c) Show a simulation of the Merge Sort algorithm to organize the whole array in increasing order. **05**

Solution:

a) **Idea 1:** Call binary search on arr[0: i] and arr[i: len(arr)] separately

```
#----------Given------------
a = [9, 12, 15, 2, 4, 5, 7, 8]
i = 3
key = 7
#---------------------------
def bin_search(arr, low, high, key):
    if low > high: return -1
    mid = (low+high)//2
    if arr[mid] == key:
        return mid
    elif key > arr[mid]:
        return bin_search(arr, mid+1, high, key)
    return bin_search(arr, low, mid-1, key)

left = bin_search(a, 0, i-1, key)
right = bin_search(a, i, len(a)-1, key)

if left < 0 and right < 0:
    print('not found')
elif left > -1:
    print(left)
else:
    print(right)
```

Idea 2: if key < arr[len(arr)-1] : just search the right portion, otherwise the left portion

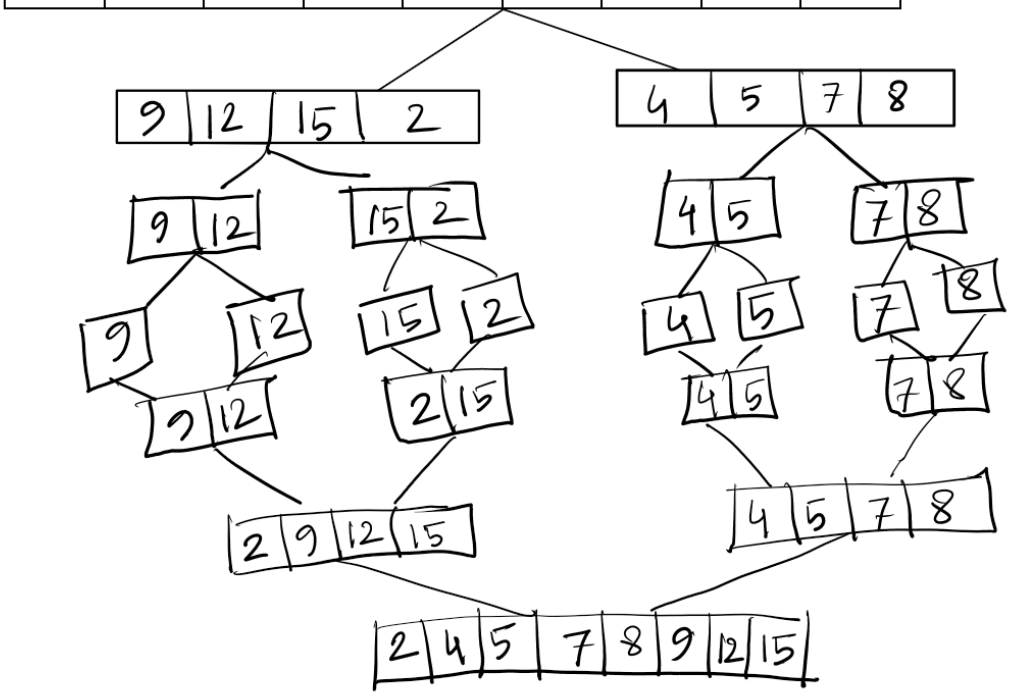
```
#----------Given------------
a = [9, 12, 15, 2, 4, 5, 7, 8]
i = 3
key = 7
#---------------------------
def bin_search(arr, low, high, key):
    if low > high: return -1
    mid = (low+high)//2
    if arr[mid] == key:
        return mid
    elif key > arr[mid]:
        return bin_search(arr, mid+1, high, key)
    return bin_search(arr, low, mid-1, key)

if key <= a[len(a)-1]:
    print(bin_search(a, i, len(a)-1, key))
else:
    print(bin_search(a, 0, i-1, key))
```

b) O(logn)

c)

index	0	1	2	3	4	5	6	7
value	9	12	15	2	4	5	7	8



3.	**CO1**	Your friend gave you a binary string B (meaning each character is either **0** or **1**). He wanted to find out how to calculate the maximum number of consecutive 0s in that particular string. For example,	

String: 100100000111	Maximum consecutive 0s: 5
String: 1010101010101	Maximum consecutive 0s: 1

You, as an algorithm enthusiast, know that this can be solved in linear time. However, your friend asked you to propose a Divide and Conquer approach

a) Name a suitable Divide and Conquer algorithm for this task. **02**
b) Explain how you can apply that algorithm in this scenario. Present your idea in a pseudocode/programmable code/Flowchart/step-by-step instructions. **06**
c) Write the time complexity of your algorithm. **02**

Solution:

a. Suitable divide and conquer algorithm for this task will be the "maximum sub array sum problem" which uses the idea of the solution existing either on the left divided subproblems, right divided subproblems or the solutions existing in the crosspoint of those two divided subproblems each.

b. Code Solution:

```
def crossmax(st,mid):
  left_sum = right_sum= 0
  i = mid
    #calculating the left portion's maximum longest sum in
crossover point
  while i>=0:
    if st[i]!="0":
      break
    left_sum+=1
    i-=1
    #calculating the right portion's maximum longest sum in
crossover point
  i=mid+1
  while i<len(st):
    if st[i]!="0":
      break
    right_sum+=1
    i+=1
  if right_sum==0 or left_sum==0:
    return float("-inf")
  return left_sum + right_sum

def maxZero(st):
  if len(st)==1:
    if st[0]=="0":
      return 1
    else: return float("-inf")
```

```
  mid = len(st)//2
  left_max = maxZero(st[:mid])
  right_max = maxZero(st[mid:])
  cross_max= crossmax(st,mid)
  return max(left_max,right_max,cross_max)

print(maxZero("100100000111"))
```

Step by step solution:

1. Make a function to identify the cross max number of 0s in a string given the crossover point at mid. There must be at least 1 0s in bith sides of the mid point or there will not be a valid solution since we are dealing with crossover events only in this step
2. Recursively divide the string into two parts by partitioning/slicing it through the mid point of the string making one string consisting of left half of the mid and one consisting the right half.
3. Set up the base case where if the length of the string reaches 1, it can either be a "0" or "1". If 0 then return value 1 as one length of consecutive 0 is found else return -inf.
4. Recursively find the left_max and right_max by implementing step 2
5. Find the cross_max by implementing the step 1 for the string given the midpoint
6. Return the max value between (left_max ,right_max, cross_max)

c. **The time complexity: T(n) = 2* T(n/2) + n = O(n logn)**
Here, in each recrusvie function cross_max can take upto n times since it can go from mid to the leftmost point and mid to rightmost point. Neglecting other constant operations, we are taking n for instruction per recursive function call.

4. **a. CO1** — **06**

You have the following adjacency matrix for a graph. However, some of the entries are missing. Your job is to find these missing entries with the help of some clues. Then draw the graph.

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	1	0	1	0	
D	0		0	0	0	0
E	0	1	1	1	0	0
F	0	0		0		0

Clues:

- E can not be reached from B
- D can be reached from A
- 2 x |E| = 3 x |V|
  (twice the no. of edges is equal to thrice the no. of vertices)
- In-degree of B is not a prime number

Solution:
(C→F : 0)
(D→B : 1)
(F→C : 1)
(F→E : 0)

b. CO2 — **04**

Is an adjacency list more memory efficient than an adjacency matrix?
Justify your reasoning with respect to directed vs undirected, weighted vs unweighted, sparse vs dense graphs.

		Solution: Directed vs Undirected: adj list may take less memory than matrix for undirected graphs. Weighted vs Unweighted: adj matrix only has to store boolean values for unweighted graphs, hence taking less memory. Sparse vs Dense: for sparse graphs, there will be a lot of 0s in the matrix representation, taking extra memory.	

B

BRAC UNIVERSITY
Department of Computer Science and Engineering

Examination: Mid Semester Exam | Semester: Fall 2023
Duration: 1 Hour 15 Minutes | Full Marks: 40

CSE 221: Algorithms

Answer the following questions.
Figures in the right margin indicate marks.

Name:	ID:	Section:

1. **a. CO2** Explain the time complexity of the following code snippet in regards of the Big-O notation: **04**

```
1. for (i=0; i<n; i+=4) {
2.       for (j=1; j<n; j*=2) {
3.             for (k=0; k<20; k++) {
4.                   print("Am I still not 30?!!");
5.             }
6.             print("Why, God, why? We had a Deal!");
7.             for (m=n; m>0; m-=4) {
8.                   print("Could you BE more dramatic?");
9.             }
10.      }
11. }
```

Solution: $n^2 * \log_2 n$

b. CO2 Consider the following functions.

$f_1(n) = (\log n)^{2000}$

$f_2(n) = n^3 \log_n(n^n)$

$f_3(n) = n^3 + 7n^2$

$f_4(n) = 4^n$

$f_5(n) = n \log n$

$f_6(n) = n * \sqrt{n}$

Now do the followings:

a. Write a correct asymptotic upper bound for each of the above. **03**
b. Sort the functions in ascending order of their growth rate, assuming n is significantly large. Just write the sorted order, no need to show any simulation. **03**

		Solution: $f_1 < f_5 < f_6 < f_3 < f_2 < f_4$	
2.	**CO3**	Consider an array containing **N** unique values where for some index **i**, the values are in increasing order from index **0** to (**i-1**), and then again from **i** to **(N-1)**. An example array is given below. Here **i=4**, it means the values are in increasing order from index 0 to 3, and then again from 4 to 7. index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 value | 9 | 12 | 15 | 20 | 4 | 5 | 7 | 8 Given the value of **i**, propose an algorithm to search a *key_value* in the array. Complexity of your algorithm must be less than **O(N)**. a) Present your solution idea as a code/ pseudocode/ flowchart/ step-by-step instructions/ logical explanations in short. b) Write the time complexity of your presented solution. c) Show a simulation of the Merge Sort algorithm to organize the whole array in increasing order.	**04** **01** **05**
3.	**CO1**	Your friend gave you a binary string B (meaning each character is either **0** or **1**). He wanted to find out how to calculate the maximum number of consecutive 0s in that particular string. For example, String: 1001000000111 | Maximum consecutive 0s: 6 String: 10101010100101 | Maximum consecutive 0s: 2 You, as an algorithm enthusiast, know that this can be solved in linear time. However, your friend asked you to propose a Divide and Conquer approach d) Name a suitable Divide and Conquer algorithm for this task. e) Explain how you can apply that algorithm in this scenario. Present your idea in a pseudocode/programmable code/Flowchart/step-by-step instructions. f) Write the time complexity of your algorithm.	**02** **06** **02**
4.	**a. CO1**	You have the following adjacency matrix for a graph. However, some of the entries are missing. Your job is to find these missing entries with the help of some clues. Then draw the graph. $\begin{array}{c\|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 0 & & 0 & 0 & 0 \\ B & 1 & 0 & 1 & 0 & 0 & 1 \\ C & 0 & 0 & 0 & 0 & 1 & 0 \\ D & 0 & 0 & 1 & 0 & 0 & 0 \\ E & 0 & & 0 & 0 & 0 & \\ F & 1 & 0 & 1 & 0 & & 0 \end{array}$ Clues: • B can not be reached from C • A can be reached from D • 2 x |E| = 3 x |V| (twice the no. of edges is equal to thrice the no. of vertices) • In-degree of C is not a prime number.	**06**

		Solution: (A→C : 1) (E→B : 0) (E→F : 1) (F→E : 0)	
	b. **CO2**	Is an adjacency list more memory efficient than an adjacency matrix? Justify your reasoning with respect to directed vs undirected, weighted vs unweighted, sparse vs dense graphs.	**04**