# National University of Computer and Emerging Sciences

**FAST School of Computing** **Fall 2019** **Islamabad Campus**

# CS-218: Data Structures

Serial No:

# Midterm Exam

**Total Time: 2 Hours**

**Total Marks: 120**

Wednesday, 16th October, 2019

## Course Instructors

Muhammad Adnan Tariq, Saba Rasheed Malik, Bilal Khalid.

_____

Signature of Invigilator

| Student Name | Roll No | Section | Signature |
|---|---|---|---|

**DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.**

**Instructions:**

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work. Use the back of the last page for rough work.
3. If you need more space write on the back side of the paper and clearly mark question and part number etc.
4. After asked to commence the exam, please verify that you have twenty three (23) different printed pages including this title page. There are a total of 3 questions.
5. Calculator sharing is strictly prohibited.
6. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

| | Q-1 | Q-2 | Q-3 | Total |
|---|---|---|---|---|
| **Marks Obtained** | | | | |
| **Total Marks** | **50** | **50** | **20** | **120** |

**Question 1 [50Marks]**

1. Complete the following code snippet that implements "bubble sort" algorithm on a linked list. [13 marks]

```
01: void BubbleSort( Node* head) {
02:    boolean exchanges;
03:    Node* iNode = Head;

04:    cNode = _____________;

05:    pNode = _____________;

06:    do {
07:        exchanges = false;

08:        while ( _________________ && cNode->next != ________ ){

09:            iNode = iNode->next;
10:            if ( cNode->value < iNode->value  ) {
11:                int tmp = cNode->value;
12:                cNode->value = iNode->value;
13:                iNode->value = tmp;
14:                exchanges = true;
15:            } // end if statement
16:        } // end while statement

17:        ___________________________

18:        ___________________________

19:    } while (exchanges); // end do while statement
20: }
```

a. Line 04: `cNode = NULL;`
   Line 05: `pNode = NULL;`
   Line 08: `cNode = iNode && cNode->next != pNode`
   Line 17: `pNode = iNode;`
   Line 18: `iNode = head;`

b. Line 04: `cNode = NULL;`
   Line 05: `pNode = Head;`
   Line 08: `cNode = iNode && cNode->next != pNode`
   Line 17: `pNode = iNode;`
   Line 18: `iNode = cNode;`

c. Line 04: `cNode = NULL;`
   Line 05: `pNode = NULL;`
   Line 08: `cNode == iNode && cNode->next != pNode`
   Line 17: `pNode = iNode;`
   Line 18: `iNode = Head;`

d. Line 04: `cNode = NULL;`
   Line 05: `pNode = Head;`
   Line 08: `cNode == iNode && cNode->next != pNode`
   Line 17: `pNode = iNode;`
   Line 18: `iNode = Head;`

e. None of the above

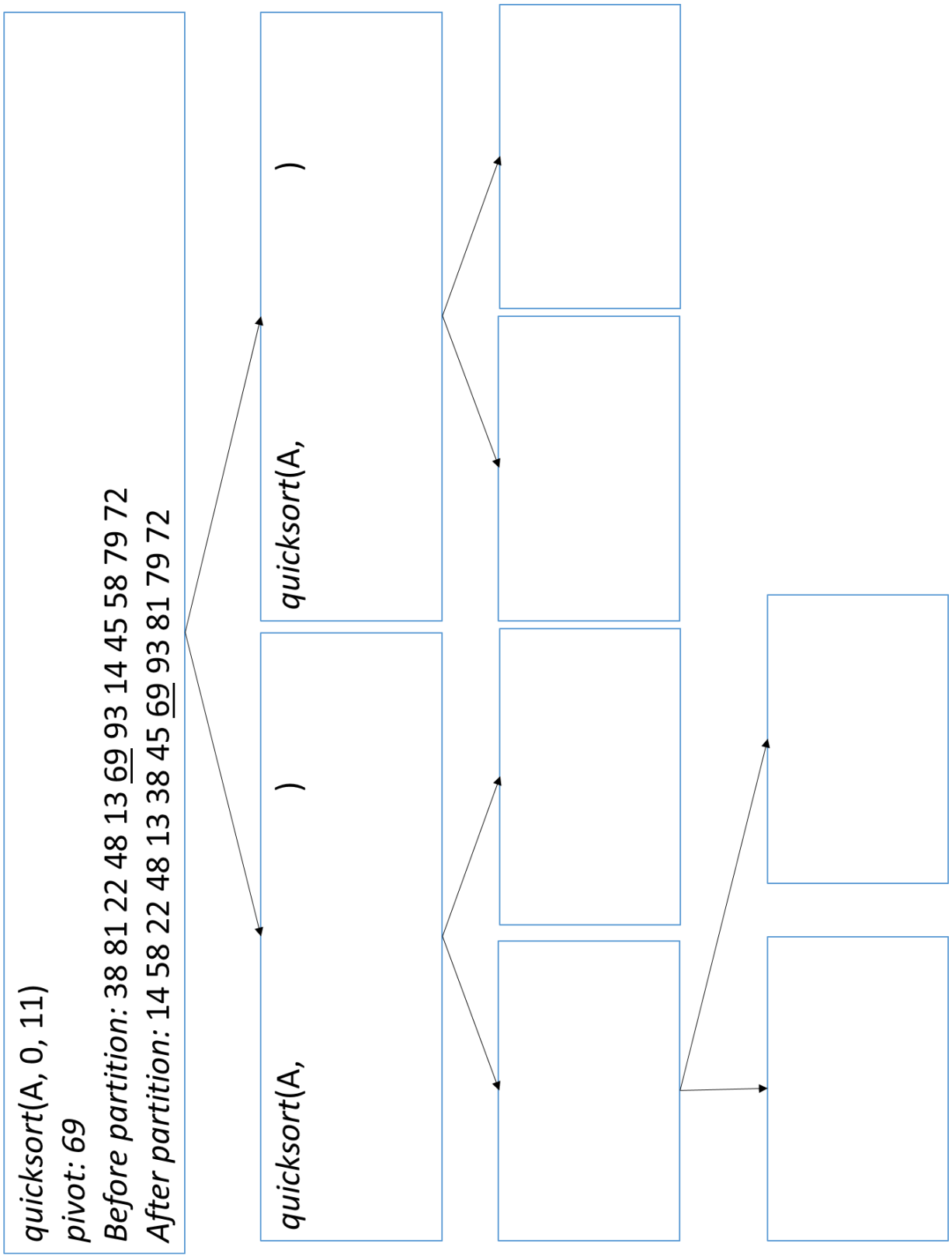
f. Bubble sort can only work on arrays

Space for rough work:

2. Consider the following code snippet to sort an array of integer values. Perform dry run of the sorting algorithm and at each recursion level, i) specify the values of parameters left and right with which the `quicksort` function is invoked, ii) specify the value of `pivot` variable, iii) update the **sub-arrays** before and after the execution of `partition` function. Empty spaces are provided on the next page. Kindly note that marks are awarded based on the correct order of elements at each level of recursion. [15 marks]

```
int partition( int A[], int left, int right){
    m = floor(left + right / 2);
    swap( A[left], A[m]);
    pivot = A[left];
    lo = left+1; hi = right;
    while ( lo ≤ hi ){
        while ( A[hi] > pivot )
            hi = hi – 1;
        while ( lo ≤ hi && A[lo] ≤ pivot )
            lo = lo + 1;
        if ( lo ≤ hi ){
            swap( A[lo], A[hi]);
            lo = lo + 1; hi = hi – 1;
        }
    }
    swap( A[left], A[hi]);
    return hi;
}
void quicksort( int A[], int left, int right){
    if ( left < right ) {
        q = partition( A, left, right);
        quicksort( A, left, q–1);
        quicksort( A, q+1, right);
    }
}
void swap (int &a, int &b){
    int temp = b;
    b = a;
    a = temp;
}
int main( ){

    int A [] = {38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72};
    quicksort (A, 0, 11);
    return 0;
}
```


quicksort(A, 0, 11)
pivot: 69
Before partition: 38 81 22 48 13 69 93 14 45 58 79 72
After partition: 14 58 22 48 13 38 45 69 93 81 79 72
quicksort(A, )
quicksort(A, )

3. Given a linked list with the following items: 1 → 2 → 3 → 3 → 4 → 6 → 8 → 5 → NULL. What will be the output of the following function? [12 marks]

```
struct node* Function (struct node* head, struct node* prev)
{
    if (head == NULL)
        return NULL;

    struct node* temp;
    struct node* curr;
    curr = head;

    while (curr != NULL && curr->data % 2 == 0) {
        temp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = temp;
    }

    if (curr != head) {

        head->next = curr;
        curr = Function(curr, NULL);
        return prev;
    }

    else {
        head->next = Function(head->next, head);
        return head;
    }
}

void main (){
    . . .
    head = Function (head, NULL);
    . . .
}
```

a. 1 → 8 → 3 → 3 → 6 → 4 → 2 → 5 → NULL (with head pointing to element 1)
b. 1 → 8 → 3 → 3 → 6 → 4 → 2 → 5 → NULL (with head pointing to element 8)
c. 1 → 2 → 3 → 3 → 8 → 6 → 4 → 5 → NULL (with head pointing to element 1)
d. 1 → 2 → 3 → 3 → 8 → 6 → 4 → 5 → NULL (with head pointing to element 2)
e. 1 → 2 → 3 → 3 → 4 → 6 → 8 → 5 → NULL (with head pointing to element 1)
f. None of above

g. Segmentation fault

Space for rough work:

4. Complete the following code that counts the duplicates in a given linked list, e.g., given a linked list 5 → 7 → 5 → 1 → 7 → NULL, the function will return 2. [10 marks]

```
01: struct Node {
02:    int data;
03:    Node* next;
04: };

05: int countNode(Node* head)
06: {
07:     int count = 0;

08:     while (_____________________________) {

09:         _____________________________________;

10:         while (ptr != NULL) {

11:             if (_____________________________) {

12:                 count++;
13:                 break;
14:             }
15:             ptr = ptr->next;
16:         }

17:         head = head->next;
18:     }
19:     return count;
20: }
```

a. Line 08: `head ! = NULL;`
   Line 09: `Node *ptr = head->next;`
   Line 11: `head->data == ptr->data`

b. Line 08: `head! = NULL;`
   Line 09: `Node *ptr = head;`
   Line 11: `head->data == ptr->data`

c. Line 08: `head->next ! = NULL;`
   Line 09: `Node *ptr = head->next;`
   Line 11: `head->data == ptr->data`

d. Line 08: `head->next ! = NULL;`
   Line 09: `Node *ptr = NULL;`
   Line 11: `head->data == ptr->next->data`

e. None of above

Space for rough work:

**Question 2 [50Marks]**

1. The **longest common subsequence (LCS)** problem is the problem of finding the longest subsequence common to all sequences in a set of sequences. Provide the dry run for below given algorithm and also find the time complexity. [8 + 2 marks]

```
#include <iostream>
using namespace std;

int max(int a, int b);

int lcs(char* X, char* Y, int m, int n)
{
	int** L = new int*[m + 1];
	for(int i = 0; i < (m+1); ++i)
		L[i] = new int[n+1];

	int i, j;
	for (i = 0; i <= m; i++) {
		for (j = 0; j <= n; j++) {
			if (i == 0 || j == 0)
				L[i][j] = 0;

			else if (X[i - 1] == Y[j - 1])
				L[i][j] = L[i - 1][j - 1] + 1;

			else
				L[i][j] = max(L[i - 1][j], L[i][j - 1]);
		}
	}
	return L[m][n];
}
int max(int a, int b)
{
	return (a > b) ? a : b;
}
int main()
{
	char X[] = "I am Fastian";
	char Y[] = "Student of Data Structure";

	int m = strlen(X);
	int n = strlen(Y);

	cout<<"Length of LCS is %d\n"<< lcs(X, Y, m, n));
	system("pause");
	return 0;
}
```

a. Use the following table for the dry run.

| | I | A | M | F | A | S | T | I | A | N |
|---|---|---|---|---|---|---|---|---|---|---|
| S | | | | | | | | | | |
| T | | | | | | | | | | |
| U | | | | | | | | | | |
| D | | | | | | | | | | |
| E | | | | | | | | | | |
| N | | | | | | | | | | |
| T | | | | | | | | | | |
| O | | | | | | | | | | |
| F | | | | | | | | | | |
| D | | | | | | | | | | |
| A | | | | | | | | | | |
| T | | | | | | | | | | |
| A | | | | | | | | | | |
| S | | | | | | | | | | |
| T | | | | | | | | | | |
| R | | | | | | | | | | |
| U | | | | | | | | | | |
| C | | | | | | | | | | |
| T | | | | | | | | | | |
| U | | | | | | | | | | |
| R | | | | | | | | | | |
| E | | | | | | | | | | |

b. Complexity of the algorithm:

2. Consider Insertion-Sort and Selection-Sort. For each algorithm, what will be the worst case asymptotic upper bound on the running time if you know additionally that?

    a. The input is already sorted?

    b. The input is reversely sorted?

    c. The input is a list containing n copies of the same number?

    For each case and each sorting algorithm, state your answer and justify it in one sentence. [6 marks]

3. Provide analysis of the given code snippet. [10 marks]

   a. In a competition, four different functions are observed. All the functions use a single for loop and within the, for loop, same set of statements are executed. Consider the following for loops:

   A) `for(i = 0; i < n; i++)`

   B) `for(i = 0; i < n; i += 2)`

   C) `for(i = 1; i < n; i *= 2)`

   D) `for(i = n; i > -1; i /= 2)`

   If n is the size of input (positive), which function is most efficient (if the task to be performed is not an issue)?

   b. Analysis of the following code snippet:

```
void silly(int n)
{
      for (int i = 0; i < n; ++i)
      {
            for (int j = 0; j < i; ++j)
            {
                  Cout<< “j = ” << j;
            }
            for (int k = 0; k < n * 3; ++k)
            {
                  Cout<< “k = ” << k;
            }
      }
}
```

c. Analysis of the following code snippet:

```
void silly(int n, int x, int y) {
      for (int i = 0; i < n; ++i) {
            if (x < y)
                  for (int k = 0; k < n * n; ++k) {
                        Cout<< “k = ” << k;
                  }
            else
                  Cout<< “i = ” << i;
      }
}
```

d. Analysis of the following code snippet:

```
void silly(int n) {
      if (n <= 0) return;
      Cout<< “n = ” << n;
      silly(n - 1);
}
```

e. Analysis of the following code snippet:

```
void silly(int n) {
      if (n <= 0) return;
      Cout<< “n = ” << n;
      silly(n / 2);
}
```

4. Consider the below given method, and provide the dry run. [8 +2 marks]

```
#include <iostream>
using namespace std;

int foo(int arr[], int n)
{
	int low = 0, high = n - 1;

	while (low <= high)
	{
		if (arr[low] <= arr[high])
			return low;

		int mid = (low + high) / 2;
		int next = (mid + 1) % n;
		int prev = (mid - 1 + n) % n;

		if (arr[mid] <= arr[next] && arr[mid] <= arr[prev])
			return mid;

		else if (arr[mid] <= arr[high])
			high = mid - 1;

		else if (arr[mid] >= arr[low])
			low = mid + 1;
	}

	return -1;
}

// main function
int main(void)
{
	int arr[] = { 5, 6, 7, 8, 9, 10, 1, 2, 3, 4 };
	int n = sizeof(arr) / sizeof(arr[0]);

	int count = foo(arr, n);
	cout<<"The array is rotated "<<count<<" times";
	system("pause");
	return 0;
}
```

| | Low | high |
|---|---|---|
| Iteration 1 | | |
| Iteration 2 | | |
| Iteration 3 | | |
| Iteration 4 | | |
| Iteration 5 | | |
| Iteration 6 | | |
| Iteration 7 | | |
| Iteration 8 | | |
| Iteration 9 | | |
| Iteration 10 | | |

a. Determine the number of iterations required to arrange the data.

b. Determine the time complexity for the function `foo()`

5. Below given a function "Foo" for the searching of an element using binary search. [7 + 7 marks]

```
int foo(int arr[], int N, int x)
{
      int low = 0, high = N - 1;
      int result = -1;

      while (low <= high)
      {
            int mid = (low + high) / 2;

            if (x == arr[mid])
            {
                  result = mid;
                  high = mid - 1;
            }

            else if (x < arr[mid])
                  high = mid - 1;
            else
                  low = mid + 1;
      }
      return result;
}
```

a. Provide dry run for the following values:

```
arr[] = { 2, 5, 5, 5, 6, 6, 8, 9, 9, 9 }
n=10
Target=5
```

| | Low | high |
|---|---|---|
| Iteration 1 | | |
| Iteration 2 | | |
| Iteration 3 | | |
| Iteration 4 | | |
| Iteration 5 | | |
| Iteration 6 | | |
| Iteration 7 | | |

| | | |
|---|---|---|
| Iteration 8 | | |
| Iteration 9 | | |
| Iteration 10 | | |

Value of result:

b. Provided another version of "foo" for the following values.

```
arr[] = { 2, 5, 5, 5, 6, 6, 8, 9, 9, 9 }
n=10
Target=5
```

```
int foo(int A[], int N, int x)
{
      int low = 0, high = N - 1;
      int result = -1;

      while (low <= high)
      {
            int mid = (low + high) / 2;
            if (x == A[mid])
            {
                  result = mid;
                  low = mid + 1;
            }

            else if (x < A[mid])
                  high = mid - 1;

            else
                  low = mid + 1;
      }
      return result;
}
```

| | Low | high |
|---|---|---|
| Iteration 1 | | |
| Iteration 2 | | |
| Iteration 3 | | |
| Iteration 4 | | |
| Iteration 5 | | |
| Iteration 6 | | |
| Iteration 7 | | |
| Iteration 8 | | |
| Iteration 9 | | |
| Iteration 10 | | |

**Question 3 [20Marks]**

1. In many fields of applied computer science (bioinformatics, optimization,), there is a need for a model of a fixed size memory where new data replace old ones (As in a queue). For example, a memory with a capacity of 7 would yield:

```
M = create-memroy(7)
store(M, 1)
store(M, 2)
store(M, 3)
store(M, 4)
store(M, 5)
store(M, 5)
store(M, 7)
store(M, 8)
print-memory(M)
>>> 2, 3, 4, 5, 5, 7, 8 //The order is preserved
```

   What kind of data structures would be adequate for this application (we would like fast access to the elements through their index)? Please elaborate your answer! [2 marks]

2. Suppose that Q is an initially empty array-based queue of size 5. Show the values of the data members front and back after each statement has been executed. Indicate any errors that might occur. [5 marks]

| | |
|---|---|
| `Queue<Character> Q( 5 );` | front = _____ back = _____ |
| `Q.enqueue ( 'A' );` | front = _____ back = _____ |
| `Q.enqueue ( 'B' );` | front = _____ back = _____ |
| `char c = Q.dequeue( );` | front = _____ back = _____ |
| `Q.enqueue ( 'C' );` | front = _____ back = _____ |
| `c = Q.dequeue( );` | front = _____ back = _____ |
| `Q.enqueue ( 'A' );` | front = _____ back = _____ |

   List of errors:

3. Write a method public void `removeAfter(Object ob)` that could be added to `ArrayQueue`: The executor removes all values from the queue that come after the element closest to the rear that is equal to the parameter. If none are equal, leave the queue as it was originally. [4 marks]

4. Given the following queue (array implementation), containing the numbers 4, 3, 6, 8 and 9. [3 marks]

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 3 | 6 | 8 | 9 | | | | | | | |

   a. What are the values of front and rear?

b. Given this queue, suppose we call Dequeue twice and Enqueue once. What would be the new values of front and rear?

c. How many elements can this queue hold?

| | | |
|---|---|---|
| (i) | Value of Front: | Value of Rear: |
| (ii) | Value of Front: | Value of Rear: |
| (iii) | No. of elements this Queue can hold: | |

5. Write a function in C++ to insert in a static circular Queue containing Players information (represented with the help of array of structure PLAYER). [3 marks]

```
struct NODE
{
   int PID;
   char Pname[20];
   NODE *Next;
};
```

6. Write a function in QUEINS() in C++ to delete a dynamically allocated Queue containing nodes of following given structure: [3 marks]

```
struct NODE
{
   int PID;
   char Pname[20];
   NODE *Next;
};
```