Concordia University	Concordia University **COMP352 - Midterm Exam – Fall 2016** Instructor: Dr. Aiman Hanna Time: 60 Minutes

Student Last Name:	**Student ID:**
First Name:	

- **Keep your answer very organized & clean. Handwriting must be readable!**
- **Exam will be marked out of 20. Exam has 8 pages.**
- **Exam is closed-book. ENCS calculators allowed.**

This section is reserved for markers					
Q1/10	**Q2/3**	**Q3/2**	**Q4/3**	**Q5/4**	**Total/20**

Question # 1 (8 marks)

For each of the following questions, **circle only one** answer in the table below. You must choose the correct answer, or the best answer if you believe multiple answers are correct. Multiple selections will void your answer. An incorrect answer may cost you ¼ of the mark (i.e. -0.25 for a question of 1 mark). **Answers MUST be recorded in that table below**. ALL OTHER MARKINGS ON THE QUESTION ITSELF, OR ELSEWHERE, WILL BE TOTALLY DISCARDED.

Q1-i	**A**	**B**	**C**	**D**	**E**		**Q1-vi**	**A**	**B**	**C**	**D**	**E**
Q1-ii	**A**	**B**	**C**	**D**	**E**		**Q1-vii**	**A**	**B**	**C**	**D**	**E**
Q1-iii	**A**	**B**	**C**	**D**	**E**		**Q1-viii**	**A**	**B**	**C**	**D**	**E**
Q1-iv	**A**	**B**	**C**	**D**	**E**		**Q1-ix**	**A**	**B**	**C**	**D**	**E**
Q1-v	**A**	**B**	**C**	**D**	**E**		**Q1-x**	**A**	**B**	**C**	**D**	**E**

i) Assume a Sequence ADT is implemented using a singular linked list that has two pointers, *head* and *tail*, pointing respectively to the first and last node of the list. Under this configurations, the Node List methods *addBefore(p, e)*, and *addAfter(p, e)*, where *p* is a position and *e* is the element to be added will result in:

A. *addBefore(p, e)* will be O(1), and *addAfter(p, e)* will be O(1)

B. *addBefore(p, e)* will be O(n), and *addAfter(p, e)* will be O(1)

C. *addBefore(p, e)* will be O(1), and *addAfter(p, e)* will be O(n)

D. *addBefore(p, e)* will be O(n), and *addAfter(p, e)* will be O(n)

E. *addAfter(p, e)* will be O(1), but it is not possible to implement *addBefore(p, e)*.

ii) Assume a Sequence ADT is implemented using a circular arrav of positions, which consequently point the elements. Under this configurations, the D.Q. operations (*addFirst(), addLast(), removeFirst()* and *removeLast()*), will result in the following complexity:

A. All 4 operations will be O(1)

B. All 4 operations will be O(n)

C. All 4 operations will be $O(n^2)$

D. *addFirst(),* and *addLast()* will be O(n). *removeFirst()* and *removeLast()* will be O(1)

E. None of the above answers is correct.

iii) Assume the following function where $n = 0, 1, 2, 3, \ldots$

$$f(n) = \frac{(n^9 + 100) * \log (n + 5) + (((4000n^7 + 1) + 100n^2) * \log n)}{n^5 + 25\, n^4}$$

In terms of Big-O approximations, the above function is:

A. $O(n^2 \log n)$

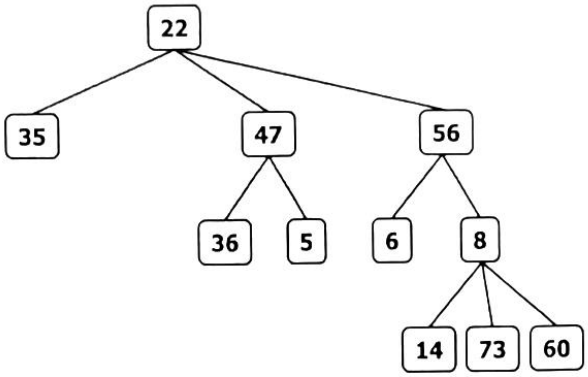
B. $O(n^4 \log n)$

C. $O(n^9 \log n)$

D. $O(n^{18} \log n)$

E. None of the above.

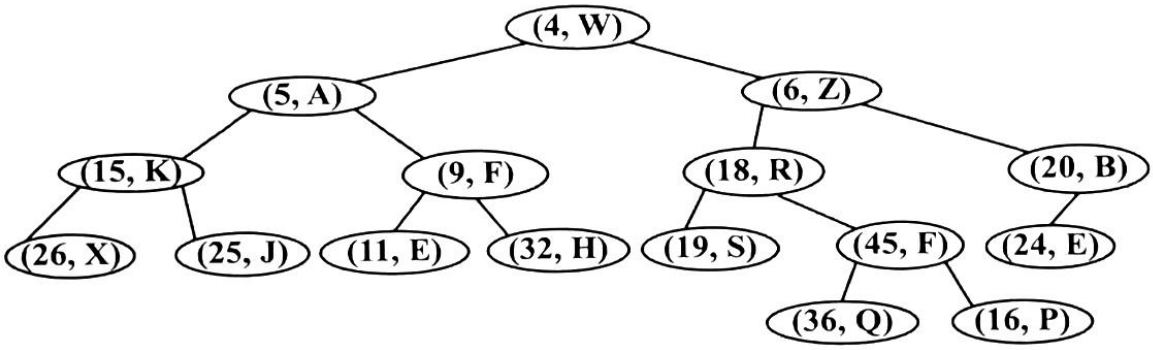
iv) Consider the following tree:



Which of the following does correctly correspond to a *postorder* traversal of the tree?

A. 22, 35, 47, 36, 5, 6, 56, 14, 73, 60, 8
B. 35, 22, 36, 47, 5, 6, 56, 14, 73, 8, 60
C. 22, 35, 47, 56, 36, 5, 6, 8, 14, 73, 60
D. 35, 36, 5, 47, 6, 14, 73, 60, 8, 56, 22
E. None of the above.

v) Given the following *incorrect* heap:



A. The heap can be corrected by removing entries (36, Q) and (16, P)
B. The heap can be corrected by moving entry (36, Q) to become the child of (26, X), and removing (16, P)
C. The heap can be corrected by moving (16, P) to become the child of (26, X), and removing (36, Q)
D. Two of the above solutions are correct.
E. None of the above solutions will be sufficient to correct the heap.

vi) Consider a circular array contains integers that are sorted in descending order. Additionally, consider a linear search algorithm that randomly decides to search the array either starting from its start (index 0) or from its end (index n-1). The *worst* case of this search algorithm for a value x occurs when:

A. x is at either end of the array
B. x is in the middle of the array
C. x is not in the array at all, or it is located at either end
D. x is smaller than the value located at index 0
E. x is larger than the value located at index n-1

vii) Consider an ADT has the following methods:

- insert(e): Inserts element e at the start of the collection
- remove(e): Deletes element e from the collection
- search(e): Searches for an element e in the collection

Additionally, it is expected that insertions in the list will be massive (done quite often), where removal and search are to be done rarely. Further, the ADT is expected to be employed in system with very critical space limitations (space is very limited; and hence space complexity is a priority). Under these conditions:

A. It is better to use expandable *non-circular arrays* as the underlying data structure of this ADT
B. It is better to use expandable *circular arrays* as the underlying data structure of this ADT
C. It is better to use *heaps* as the underlying data structure of this ADT
D. It is better to use *singular linked lists* underlying data structure of this ADT
E. It is better to use *doubly linked lists* underlying data structure of this ADT

viii) Consider the use of binary recursion by a method called *multiplyArr(A, i, n)*, which accepts an array of integers, A, a starting index, i, and a number of elements, n. The method computes the total value (the product) resulting from multiplying all n elements of the array starting from index i. Precisely, the method returns A[i] if n = 1; otherwise it recursively returns (the multiplied value of the left half of the array) * (the multiplied value of the right half of the array). If n is odd, then the left half is larger by one element. The number of needed calls for the method to finish will result in:

A. Time complexity of $O(n^2)$ and stack growth of $O(n^2)$.
B. Time complexity of $O(n^2)$ and stack growth of O(n).
C. Time complexity of O(n) and stack growth of O(log n).
D. Exponential time complexity of $O(2^n)$ but stack growth of $O(n^2)$.
E. None of the above answers is correct.

ix) Consider the following code segment, where *S1* indicates any number of statements that would execute in a constant time:

```
for (int k = n;  k >= 1;  )
    for (int i = 1; i <= 1000; )
        for (int j = 1; j <= n; )
            S1
            i = i * 2;
            j = j+2;
            k = k/4;
```

In terms of Big-O approximations, the above segment has a complexity of:

A. O($n \log n$)
B. O(n^2)
C. O(n^3)
D. O(∞); program never ends
E. None of the above.

x) Given the following algorithm:

```
Algorithm Fun(A, B):
 Input: Arrays A and B each storing n ≥ 1 integers.
 Output: Some computations over the contents of the arrays
  c ← 0
  t ← 0
 for i ← n^2  to  1 do
     s ← 0
     if (true)
        for j ← n-1 to 0 do
            j ← j / 2
     for k ← 0 to n-1 do
        s ← s * A[0]
        t ← t * 2
        for u ← 1 to k do
            s ← s / A[u]
     if B[i] = s then
        c ← c + t + s
 return c
```

In terms of Big-O approximations, the above algorithm has a complexity of:

A. O(n^2)
B. O(n^3)
C. O(n^4)
D. O($n^2 \log n$)
E. O($n^4 \log n$)

Question # 2 (3 marks)

Give a Big-Ω and Big-Θ estimates for each of the following functions. You only need to give the correct estimates; no proof is required.

A) $f(n) = (2^n + n^2)(n^5 + 5^n) + (2^n + n^2)/n$

Big-Ω is: Ω( ).

Big-Θ is Θ( ).

B) $f(n) = (n^2)(n^4 \log n + \log n + 2)(6 \log n + 49)(n^3 + 1)$

Big-Ω is: Ω( ).

Big-Θ is Θ( ).

C) $f(n) = (22\, n^n + n * 2^n + 9^n)(n! + 3^n + n^3 \log n)$

Big-Ω is: Ω( ).

Big-Θ is Θ( ).

Question # 3 (2 marks)

Indicate whether each of the statements is *true* or *false*; no explanation is required.

A) 0000001 $n^4 + 900000000\, n^3 \log n$ is $\mathbf{O}(n^7 \log n)$. □True False□

B) $3n^2 \log n + 3n^3 \log n$ is $\Omega(1)$. □True False□

C) The hight of a proper binary tree with n nodes can be larger than $\log n$.
□True False□

D) Linear recursion will always result in a space complexity of $O(n)$.
□True False□

Question # 4 (4 marks)

Given the following function: $f(n) = 5\,n^{12} + 4n^9 \log n + 36\,n^4 + 20\,n^3 + 50\,n + 50.$

i) **Prove** or **disprove** that the above function is $\Omega(n^9 \log n)$. Full and **detailed step-by-step** proof/disproof is required.

ii) Is the above function $\Theta(n^{12} \log n)$? If *no*, explain very clearly why it is not (it cannot be). If *yes*, show a **detailed step-by-step** proof of that in a similar fashion to what was done in # i above.

Question # 5 (5 marks)

Given an array of integers of size n, and a constant, c, you are required to write an algorithm that would reverse particular elements of the array, while keeping the rest of the elements intact. In particular, the algorithm needs to reverse the two elements at two far ends of the array then continues to reverse the elements that are c distance away from these ends, and so on. For example, given an array A, of $n = 100$ elements, and c as 4, the algorithm will reverse elements at indices (0 & 99), (4 & 95), (8 & 91), (12 & 87), (16 & 83), etc. Notice this is just an example and your solution must not be limited to it.

In brief, here are the requirements: **Using pseudo code you are required to write a recursive algorithm**, called ***PartialReverse***, which expects an array A of n integers and a constant c. The algorithm will then changes the contents of the array by reversing specific elements as described above. You **must give pseudo code** (not code in Java, C++, etc.)

i) *PartialReverse* Algorithm

ii) In terms of Big-O, what is the *time* complexity of your algorithm?

Answer: Time complexity is: **O(** **)**

iii) In terms of Big-O, what is the *space* complexity of your algorithm?

Answer: Space complexity is: **O(** **)**