⭐ [F24] Midterm Exam KEY 1 Multiple Choice ONLY

● Graded

Student

Qiulin Fan

Total Points

52.5 / 60 pts

Question 1

Fooing Around

2.5 / 2.5 pts

+ **0 pts** A

✔ + **2.5 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

– **0.25 pts** Blank/Multiple

Question 2

Recursive Truths

0 / 2.5 pts

✔ + **0 pts** A

+ **0 pts** B

+ **0 pts** C

+ **2.5 pts** D

+ **0 pts** E

– **0.25 pts** Blank/Multiple

Question 3

Master of Recursion

2.5 / 2.5 pts

✔ + **2.5 pts** A

+ **0 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

– **0.25 pts** Blank/Multiple

Question 4

Recurrence Relation

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

+ **0 pts** C

+ **0 pts** D

✔ + **2.5 pts** E

− **0.25 pts** Blank/Multiple

Question 5

Counting Sets

2.5 / 2.5 pts

✔ + **2.5 pts** A

+ **0 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 6

Unknown Container Library

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

✔ + **2.5 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 7

Binary Search

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

✔ + **2.5 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 8

Circular Buffer Operations

0 / 2.5 pts

+ 2.5 pts A

+ 0 pts B

+ 0 pts C

+ 0 pts D

✔ + 0 pts E

– 0.25 pts Blank/Multiple

Question 9

Stack of Exams

2.5 / 2.5 pts

+ 0 pts A

+ 0 pts B

+ 0 pts C

+ 0 pts D

✔ + 2.5 pts E

– 0.25 pts Blank/Multiple

Question 10

Sorted Containers

2.5 / 2.5 pts

+ 0 pts A

✔ + 2.5 pts B

+ 0 pts C

+ 0 pts D

– 0.25 pts Blank/Multiple

Question 11

Quicksort Facts

2.5 / 2.5 pts

+ 0 pts A

+ 0 pts B

+ 0 pts C

✔ + 2.5 pts D

+ 0 pts E

– 0.25 pts Blank/Multiple

Question 12

Blind Pivot

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

+ **0 pts** C

✔ + **2.5 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 13

Friendly Pivot

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

✔ + **2.5 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 14

Mergesort Facts

2.5 / 2.5 pts

+ **0 pts** A

✔ + **2.5 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 15

Linked-List Sorting

0 / 2.5 pts

+ **2.5 pts** A

✔ + **0 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 16

Find Asymptotic Runtime

2.5 / 2.5 pts

+ **0 pts** A

✔ + **2.5 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

– **0.25 pts** Blank/Multiple

Question 17

Using a Deque

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

+ **0 pts** C

✔ + **2.5 pts** D

+ **0 pts** E

– **0.25 pts** Blank/Multiple

Question 18

Heap Modification

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

✔ + **2.5 pts** C

+ **0 pts** D

+ **0 pts** E

– **0.25 pts** Blank/Multiple

Question 19

STL Queue

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

+ **0 pts** C

+ **0 pts** D

✔ + **2.5 pts** E

– **0.25 pts** Blank/Multiple

Question 20

Sorting a Linked List

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

+ **0 pts** C

+ **0 pts** D

✔ + **2.5 pts** E

− **0.25 pts** Blank/Multiple

Question 21

Pick a Sort

2.5 / 2.5 pts

✔ + **2.5 pts** A

+ **0 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 22

Array Intersection

2.5 / 2.5 pts

+ **0 pts** A

+ **0 pts** B

✔ + **2.5 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 23

STL Iterators

2.5 / 2.5 pts

+ **0 pts** A

✔ + **2.5 pts** B

+ **0 pts** C

+ **0 pts** D

+ **0 pts** E

− **0.25 pts** Blank/Multiple

Question 24

Binary Heaps Operations

2.5 / 2.5 pts

+ 0 pts A

+ 0 pts B

+ 0 pts C

✔ **+ 2.5 pts** D

– 0.25 pts Blank/Multiple

Question 25

Key Number

0 / 0 pts

✔ **+ 0 pts** Correct key

+ 0 pts Wrong key!

The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2024

MIDTERM EXAM
KEY 1

Thursday, October 17, 2024
7:00 PM - 9:00 PM (120 minutes)

Uniqname: rynnefan **Student ID:** 58848733

Name: Qiulin Fan

Uniqname of person to your left: 1

Uniqname of person to your right: aplong

Honor Pledge:
"I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code."

Signature: Qiulin Fan

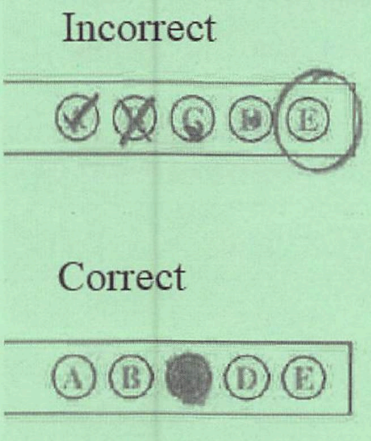
INSTRUCTIONS:

- The exam is closed book and notes except for one 8.5"x11" sheet of handwritten notes (both sides). All electronic device use is prohibited during the exam (calculators, phones, laptops, etc.).
- Print your name, student ID number, and uniqname **LEGIBLY**. Sign the Honor Pledge; this pledge applies to both the written and multiple choice sections. Your exam will not be graded without your signature.
- Record the **UNIQNAME** of the students to both your left and right. Write down `nullptr` if you are at the end of a row and there is no one on a given side.
- Record your **UNIQNAME** at the top of each odd-numbered page in the space provided. This will allow us to identify your exam if pages become separated during grading.
- This exam booklet contains **BOTH** the multiple choice and written portions of the exam. Instructions for each portion are provided at the beginning of their respective sections.
- **DO NOT** detach any pages from this booklet.
- There should be 22 pages in this book; if you are missing any pages, please let an instructor know.

Multiple Choice Portion [60 points]

MULTIPLE CHOICE INSTRUCTIONS:

- Select only **ONE** answer per multiple choice question.
- There are 24 multiple choice questions worth 2.5 points each.
- Record your choices for all multiple choice questions using the circles in the boxed area next to each question. Use a number 2 pencil, not a pen (so that you may change your answer). Fill the circle completely. There is no partial credit for multiple-choice questions. Make sure you bubble in the correct letter. See the example below for how to select your answer.



- There is no penalty for wrong answers: it is to your benefit to record an answer to each multiple-choice question, even if you're not sure what the correct answer is.
- The questions vary in difficulty — try not to spend too much time on any one question. Use your time wisely.
- When asked about memory complexity, consider all possible sources (stack and heap).
- The term "heap" as a data structure refers to a binary heap, unless explicitly stated otherwise.
- The term "binary search tree" refers to a standard implementation that does not self-balance, unless explicitly stated otherwise.

Record your answers in the bubbles next to each question name.

1. Fooing Around

(A) ● (C) (D) (E)

What is the time complexity of the function `foo()`?

```
int foo(int n, int m) {
   int totalFoos = 0;
   for(int i = 0; i < n * n; ++i) {
      for(int j = 1; j < m * m; j += j) {
         totalFoos += j;
      } // for j
   } // for i
   return totalFoos;
} // foo()
```

n²

$\log_2 m^2 = 2\log m$

A) $\Theta(n^2 \log(\sqrt{n}))$

B) $\Theta(n^2 \log(m))$ ✓

C) $\Theta(n^2 \log n))$

D) $\Theta(n^2 + \log m)$

E) $\Theta(n^2 + m^2 \log(mn))$

2. Recursive Truths

(B) ● (C) (D) (E)

Which of the following statements about recursion are **TRUE**?

i. A tail recursive algorithm can always be written iteratively, and vice versa.

ii. A tail recursive algorithm uses $O(1)$ additional stack space. ✓

iii. A recursive algorithm with runtime $T(n) = T(n-1)$ will never have better asymptotic time complexity than a recursive algorithm with runtime $T(n) = T(n/2)$. ✗

$=0$

iv. In a recursive function, the local variables are stored in heap memory and start returning from heap memory once the algorithm reaches the base case. ✗

A) i, ii

B) ii, iv

C) ii, iv

D) i, ii, iii

E) i, iii, iv

Record your answers in the bubbles next to each question name.

3. Master of Recursion ● B C D E

For which of the following recurrence relations would you **NOT** be able to use the Master Theorem to derive an asymptotic bound for the time complexity of the algorithm? Select all that apply:

- **i.** $T(n) = 4T(n/5) + n$
- **ii.** $T(n) = T(n-1) + 4n^3 + 5$
- **iii.** $T(n) = (1/2)T(n/2) + n^2$
- **iv.** $T(n) = T(n/2) + T(n/4)$
- **v.** $T(n) = 5T(n/5) + e^4$

A) ii, iii, iv

B) iii, iv

C) i, iii, v

D) i, ii, iii, iv

E) All are solvable with the Master Theorem

4. Recurrence Relation A B C D ●

What is the complexity of $T(n) = 4T(n-1) + c$ where $n > 1$ and $T(n)$ is constant for $n = 1$?

A) $O(n^2)$

B) $O(n^3)$

C) $O(n^4)$

D) $O(3^n)$

E) $O(4^n)$

5. Counting Sets ● B C D E

Consider the following union-find container. Among these 10 elements, how many disjoint sets are there?

Item	1	2	3	4	5	6	7	8	9	10
Representative	5	1	4	5	6	6	3	8	10	1

A) 2

B) 3

C) 4

D) 5

E) 6

Record your answers in the bubbles next to each question name.

6. Unknown Container Library

Ⓐ Ⓑ ● Ⓓ Ⓔ

You are given an unknown STL container that is initially empty. The following operations are called in sequence:

```
container.push(-4)
container.push(-1)
container.push(10)
container.push(9)
container.push(1)
container.pop()
container.pop()
```

stack
max PQ

If you are told that the smallest element remaining in the container is -4, then you can deduce that the unknown STL container must be a:

A) Stack or queue

B) Stack or min-priority queue

C) Stack or max-priority queue ✓

D) Queue or max-priority queue

E) Min priority queue or max-priority queue

7. Binary Search

Ⓐ Ⓑ ● Ⓓ Ⓔ

What would happen if you tried to apply binary search on an unsorted array?

A) Applying binary search on an unsorted array will always result in a time complexity of $\Theta(\log n)$.

B) The algorithm will always find the target in the unsorted array, but the index returned may be incorrect.

C) The algorithm may return an incorrect index or fail to find the target.

D) The algorithm will sort the array as it searches, ensuring accurate results.

E) None of the above

3 x
→
1 3 2 6 5

Record your answers in the bubbles next to each question name.

8. Circular Buffer Operations

Ⓐ Ⓑ Ⓒ Ⓓ ●

Assume you have a `queue<char>` implemented with a circular buffer with initial capacity of 3. If necessary, the buffer will be doubled in size (as done in lecture). The following operations are performed:

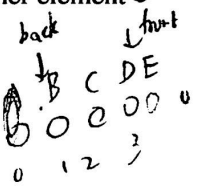
```
queue.push('A')
queue.push('B')
queue.push('C')
queue.pop()
queue.push('D')
queue.push('E')
```

Which of the following are **TRUE** once all the operations are completed?

i. The container will need to be resized in order to add another element

ii. Element `'D'` has gone from index 0 to index 2

iii. The back index of the buffer is 0



A) ii

B) i, ii

C) ii, iii

D) i, ii, iii

E) None of the above

9. Stack of Exams

Ⓐ Ⓑ Ⓒ Ⓓ ●

You are given a stack of n unsorted EECS 281 exam booklets to sort by exam score. Scores are integer numbers ranging from 0 to 100, and should be sorted in increasing order and placed back into a stack. In other words, the bottom of the stack should be the smallest scores, and the top should be the largest. Given at most $O(n)$ additional memory, what is the worst case runtime of the best possible algorithm to sort the exams?

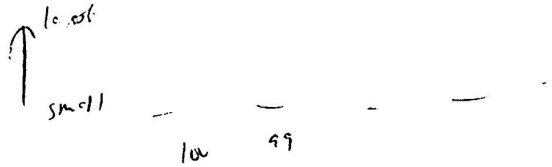
A) $\Theta(n^2)$

B) $\Theta(n \log n)$

C) $\Theta(1)$

D) $\Theta(2^n)$

E) $\Theta(n)$



Record your answers in the bubbles next to each question name.

10. Sorted Containers (A) (B) (C) (D)

Consider the following operations on a sorted vector and sorted linked list:

- Insertion of a new element
- Search for an existing element
- Deletion of an existing element

Which of the following statements correctly matches the time complexities for these operations in both data structures?

A) Insertion: Sorted Vector - $O(1)$, Sorted Linked List - $O(n)$
Search: Sorted Vector - $O(n)$, Sorted Linked List - $O(n)$
Deletion: Sorted Vector - $O(n)$, Sorted Linked List - $O(n)$

B) Insertion: Sorted Vector - $O(n)$, Sorted Linked List - $O(n)$
Search: Sorted Vector - $O(\log n)$, Sorted Linked List - $O(n)$
Deletion: Sorted Vector - $O(n)$, Sorted Linked List - $O(n)$

C) Insertion: Sorted Vector - $O(n)$, Sorted Linked List - $O(1)$
Search: Sorted Vector - $O(\log n)$, Sorted Linked List - $O(n)$
Deletion: Sorted Vector - $O(n)$, Sorted Linked List - $O(n)$

D) Insertion: Sorted Vector - $O(\log n)$, Sorted Linked List - $O(1)$
Search: Sorted Vector - $O(n)$, Sorted Linked List - $O(\log n)$
Deletion: Sorted Vector - $O(n)$,Sorted Linked List - $O(n)$

11. Quicksort Facts (A) (B) (C) (D) (E)

Which of the following statements about Quicksort is **TRUE**?

A) The quicksort algorithm, as defined in class, is stable.

B) Quicksort cannot be done in-place.

C) In the average case, the time complexity is $\Theta(n \log n)$ and space complexity is $\Theta(\log n)$ in stack frames.

D) If the pivot was always chosen (in constant time) to be the median and the array values were distinct, then the worst-case time complexity would be $\Theta(n \log n)$.

E) Quicksort cannot be implemented using iteration instead of recursion.

Record your answers in the bubbles next to each question name.

12. Blind Pivot

Ⓐ Ⓑ Ⓒ ● Ⓔ

Suppose you want to run the Quicksort implementation from lecture on the following data but you randomly select value 3 as your first pivot.

9	-2	5	3	15	-1	4	6	7

How many values would have been a **BETTER** pivot choice?

-2 -1 3 4 5 6 7 9 15

A) 0

B) 1

C) 2

D) 3 ✓

E) 4

13. Friendly Pivot

Ⓐ Ⓑ ● Ⓓ Ⓔ

Suppose your friend said he could find the median of a given data set in $O(1)$ time. If you were to use your friend's talent to optimize a Quicksort algorithm, what would be the time complexity of the algorithm?

A) Best: $O(n)$, Average: $O(n \log n)$, Worst: $O(n \log n)$

B) Best: $O(n \log n)$, Average: $O(n \log n)$, Worst: $O(n^2)$

C) Best: $O(n \log n)$, Average: $O(n \log n)$, Worst: $O(n \log n)$

$T(n) = 2T(\frac{n}{2}) + O(n)$

D) Best: $O(n)$, Average: $O(n)$, Worst: $O(n)$

E) Best: $O(\log n)$, Average: $O(n \log n)$, Worst: $O(n \log n)$

14. Mergesort Facts

Ⓐ ● Ⓒ Ⓓ Ⓔ

Which of the following is **FALSE** about top-down Mergesort?

A) It is $O(n \log n)$ ✓

B) It requires random access of elements ✗

C) It is insensitive to nearly sorted arrays ✓

D) It requires $O(n)$ extra memory ✓

E) It is stable ✓

Record your answers in the bubbles next to each question name.

15. Linked-List Sorting (A) (B) (C) (D) (E)

Which of the following sorting algorithms can sort a singly-linked list in-place with a worst-case time complexity of $\Theta(n \log n)$?

A) Mergesort

B) Quicksort

C) Selection sort

D) Insertion sort

E) The worst-case time complexity is always $\Theta(n^2)$

16. Find Asymptotic Runtime (A) (B) (C) (D) (E)

What is the asymptotic time complexity of `foo()`?

```
int bar(int m) {
    int c = 0;
    for (int i = 1; i < m; i *= 5)
        c += 3;
    return c;
} // bar()

int foo(int n, int m) {
    int x = m * m * m * m * m;
    int result = bar(x);
    int count = 0;
    while (count < m) {
        for (int i = 0; i < n; ++i)
            result += 4;

        ++count;
    } // while
    return result;
} // foo()
```

A) $\Theta(m \log m + nm)$

B) $\Theta(\log_5(m) + nm)$

C) $\Theta(n \log_5(m) + m)$

D) $\Theta(m + nm)$

E) $\Theta(\log(m^5) * nm)$

Record your answers in the bubbles next to each question name.

17. Using a Deque

Ⓐ Ⓑ Ⓒ ● Ⓔ

Which of the following containers can be implemented efficiently using a `std::deque<>` as the underlying container?

A) Stack ✓

B) Queue ✓

C) Priority queue ✓

D) All of the above

E) None of answer choices (A), (B), (C) or (D)

18. Heap Modification

Ⓐ Ⓑ ● Ⓓ Ⓔ

Assume you have a binary heap based on maximum value being highest priority, with the current contents being $\{9, 8, 7, 6, 5, 4, 3, 2\}$. The heap property should be restored after every operation, using the algorithms discussed in lecture. After each operation, perform **only** a `fixUp()` or `fixDown()` as necessary, never update all priorities.

Perform the following operations:

i. Delete the maximum element.

ii. Update the element with value 7 to have value 1.

iii. Insert a new element with value 4.

What are the heap contents now?

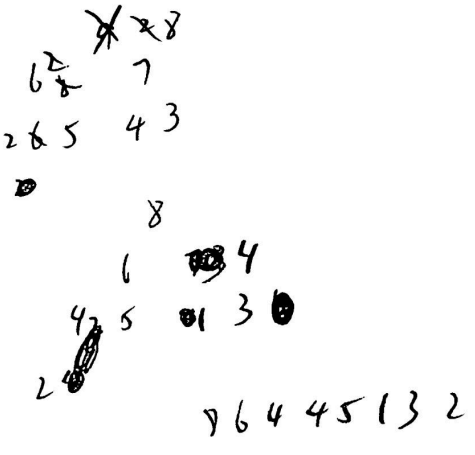
A) $\{8, 6, 4, 2, 5, 3, 1, 2\}$

B) $\{8, 4, 6, 3, 1, 5, 4, 2\}$

C) $\{8, 6, 4, 4, 5, 1, 3, 2\}$

D) $\{8, 4, 6, 4, 5, 1, 3, 2\}$

E) $\{8, 6, 5, 4, 4, 3, 1, 2\}$



Record your answers in the bubbles next to each question name.

19. STL Queue (A) (B) (C) (D) ●

Which of the following statements about the STL `std::queue` is **TRUE**?

A) The member function `pop()` can be used to access the data at the front of the queue

B) The phrase "last in, first out" describes the operation of a queue

C) The `back()` member function allows a user to look at the data at the back of a queue, though the element cannot be modified

D) The member function `push()` will add data to the front of the queue

E) All of the above statements about `std::queue` are false

20. Sorting a Linked List (A) (B) (C) (D) ●

Which of the following sorting algorithms could be implemented on a doubly-linked list **WITHOUT** making the asymptotic worst-case complexity even worse? You must perform the sorting in-place, you **CANNOT** just copy to an array and then use the normal algorithm.

i. Bubble sort

ii. Selection sort

iii. Insertion sort

iv. Quicksort

v. Heapsort

A) i, ii, and iii only

B) iv and v only

C) i and iii only

D) i and ii only

E) All except v

Record your answers in the bubbles next to each question name.

21. Pick a Sort (A) (B) (C) (D) (E)

Which of the sorts presented in class is best-suited for input data that is already nearly sorted, where many values are out of place, but every value is very close to its correct position?

A) Insertion Sort

B) Selection Sort

C) Quicksort

D) Heapsort

E) Counting Sort

22. Array Intersection (A) (B) (C) (D) (E)

Given two sorted arrays, what is the worst-case time complexity of finding the intersection of the two arrays if you use the most efficient algorithm? Let n be the size of the larger array. For example:

```
int arr1[] = {1, 2, 3, 4, 5};
int arr2[] = {1, 4, 5, 6};
```

would produce the output

```
int output[] = {1, 4, 5};
```

A) $\Theta(1)$

B) $\Theta(\log n)$

C) $\Theta(n)$

D) $\Theta(n \log n)$

E) $\Theta(n^2)$

23. STL Iterators (A) (B) (C) (D) (E)

Which of the following is **NOT** an STL iterator type?

A) bidirectional iterator

B) restricted iterator

C) forward iterator

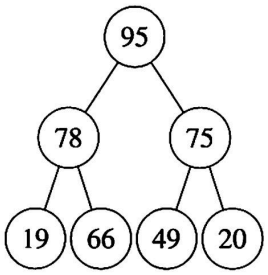
D) random access iterator

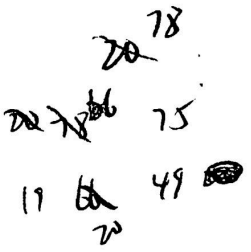
E) reverse iterator

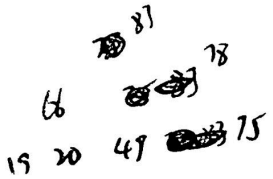
Record your answers in the bubbles next to each question name.

24. Binary Heaps Operations Ⓐ Ⓑ Ⓒ ●

Suppose we have the max-heap represented below:





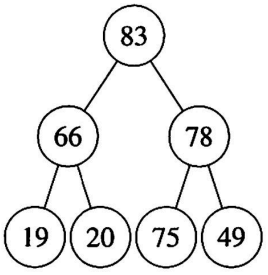


And then carry out the following operations:

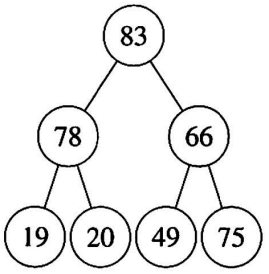
1. Pop the top value
2. Push the value 83

Which diagram represents the state of the heap after these two operations have completed?

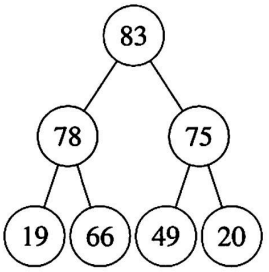
A)



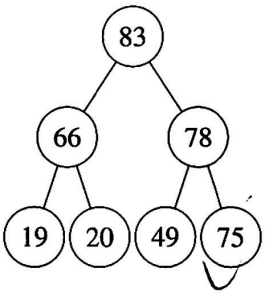
C)



B)



D)



Written Portion [40 points]

WRITTEN PORTION INSTRUCTIONS:

- There are **two** questions in this section, with the following point distribution:

Question 25	20 points
Question 26	20 points

- Please write your solution legibly in the space provided. Solutions that are difficult to read may receive fewer points.
- Use the back of the question page to work out solutions for each problem and then rewrite your final answer in the space provided.
- The directions for each problem will specify which STL algorithms/functions may be used in your solution.
- Solutions that exceed the allowed line count will lose one (1) point per line over the limit.
- Partial credit will be awarded for partial solutions.
- Errors in program logic or syntax may result in a reduced score.
- Be sure to consider all possible sources of memory (stack and heap) and time complexity.

---

25. Programming: Linear-time Algorithms and the STL [20 Points]

Suppose you have two **sorted** sequences (sorted with respect to the given functor `pred`), both containing the same data type, and want to know if the first sequence contains the second.

This function will accept two iterators into the first input range `[first1, last1)`, two iterators into the second input range `[first2, last2)`, and a functor `pred` which accepts two items and returns `true` if the first item is less than the second item. If the second sequence contains two or more copies of the same value, the first sequence would need at least the same number of copies. An empty sequence is contained within all other sequences, even another empty sequence.

Example:

```
vector<int> seq1 { 1, 2, 3, 4, 5, 6, 7 };
vector<int> seq2 { 2, 4, 6 };
bool contained = sequence_contains(seq1.begin(), seq1.end(),
                                   seq2.begin(), seq2.end(),
                                   std::less<int>());
```

After running this code, `contained` should be `true`. IF you added another copy of 2 to `seq2`, it should be `false` (because `seq1` is missing a second copy of 2). IF you *instead* added 8 to `seq2`, it should be `false` (because `seq1` does not have the value 8).

Return: A Boolean value, `true` if the second sequence was found, or `false` otherwise.

Complexity: $O(n)$ time and $O(1)$ space, where n is equal to $2*[(last1-first1)+(last2-first2)-1]$.

Implementation: Use the back of this page as a working area, then rewrite **NEATLY** on the front. Limit: 13 lines of code (points deducted if longer). You may **NOT** use anything from the STL.

```
template <class ForwardIterator1, class ForwardIterator2, class Predicate>
bool sequence_contains(ForwardIterator1 first1, ForwardIterator1 last1,
                       ForwardIterator2 first2, ForwardIterator2 last2,
                       Predicate pred) {
    if (first2 == last2) return true; // empty, then true
    for (ForwardIterator it = first1; it != first2; ++it) {
        if (!pred(*it, *first2) && !pred(*first2, *it)) //equal
            first2++;
        if (first2 == last2)
            return true;
    }
    return false;
}
```

This page is intentionally left blank.
You may use this page as working space.

26. Programming: Big-O'le Meals [20 Points]

You work for Big-O'le Meals and are tasked with designing an algorithm that assigns delivery orders to drivers, prioritizing both order delivery times and driver ratings. Each driver has an initial rating, which is updated based on their performance on each delivery. The goal is to maximize the tips received by drivers while ensuring that orders with shorter delivery times are prioritized.

Input: A vector of n orders and a vector of m drivers, where $n > m > 0$, using the following two structs,

```
struct Order {
  int deliveryTime;
  double ratingChange;
  double price;
  double tipPercent;
};
```

```
struct Driver {
  int id;
  double rating;
};
```

- `deliveryTime`: time required for the delivery.
- `ratingChange`: change in the driver's rating after completing this order.
- `price`: Price of the order.
- `tipPercent`: tip as a percentage of the price, i. e., value is between 0 and 1.0.
- `id`: unique identifier for the driver (id's are in the range $[0, m)$ where m is the number of drivers).
- `rating`: rating of the driver (can be positive or negative). Higher values indicate a better rating.

Output: A vector where the value at index i represents the total tips to be earned by driver i given the computed assignment of orders to drivers.

Example: Given the following:

- `vector<Driver> drivers = {{0, 3.1}, {1, 4.3}, {2, 3.1}, {3, 0.0}};`
- `vector<Order> orders = {{4, -2.1, 15, 0.1}, {13, -0.7, 20, 0.25}, {12, 0.5, 10, 0.5}, {16, 3.0, 8, 0.75}};`

The vector returned should be `{10, 1.5, 6, 0}` since:

- Driver 1 takes order 0 with delivery time 4 resulting in \$1.5 tips and rating decrease to 1.1. The reason this order is taken first is explained with the functor (Part 2).
- Driver 0 takes order 2 with delivery time 12 resulting in \$5 of tips and rating increase to 3.6.
- Driver 0 takes order 1 with delivery time 13 resulting in \$5 of tips and rating decrease to 2.9.
- Driver 2 takes order 3 with delivery time 16 resulting in \$6 of tips and rating increase to 6.1.

Constraints: You must process the orders in $O(n \log m)$ time and use $O(m)$ space, where n is the number of orders and m is the number of drivers.

This page is intentionally left blank.
You may use this page as working space.

Implementation: Use the back of this page as a working area, then rewrite **neatly** on the front. For these functors, you may **NOT** use anything from the STL.

Part 1: Implement Driver Comparator (2 Points)
You should return **true** if the driver object `a` should come *after* the driver object `b` and **false** otherwise.

- Drivers with higher ratings should be prioritized.
- If two drivers have the same rating, prioritize the driver with the lower `id`.

Limit: 6 lines of code (points deducted if longer).

```
struct DriverComp {
  bool operator()(const Driver &a, const Driver &b) {
        if (a.rating < b.rating) return true;
        else if (a.rating == b.rating) return (a.id > b.id);
        else return false;
  }
```

Part 2: Implement Order Comparator (3 Points)
You should return **true** if the order object `a` should come *after* the order object `b` and **false** otherwise.

- Orders with shorter delivery times should be prioritized.
- If two orders have the same delivery time, prioritize the order that has the higher expected tip. The tip is calculated as `price * tipPercent`.

Limit: 8 lines of code (points deducted if longer).

```
struct OrderComp {
  bool operator()(const Order &a, const Order &b) {
        if(a.deliveryTime < b.deliveryTime) return false;
        else if (a.deliveryTime == b.deliveryTime)
            return (a.price * a.tipPercent < b.price * b.tipPercent);
        else return true;
  }
};
```

This page is intentionally left blank.
You may use this page as working space.

Implementation: Use the back of this page as a working area, then rewrite **neatly** on the front. For this function, you **CAN** use anything from the STL.

Part 3: Deliver Orders (15 points)
Implement the function `deliverOrders()` that takes in a vector of `Order` objects and a vector of `Driver` objects; and returns a `vector<double>` corresponding to the total tips for each driver given the prioritized assignment of orders to drivers.

- Initially, there will be more orders than drivers. Start by assigning orders to the available drivers based on the priorities from `DriverComp` and `OrderComp`.
- After an order is assigned to a driver, the driver's rating is updated by the `ratingChange` of the order they delivered.
- Continue assigning new orders to drivers based on the updated ratings. At no point can the number of orders assigned exceed the number of drivers.

Limit: 20 lines of code (points deducted if longer).

```
vector<double> deliverOrders(const vector<Order> &orders,
                             const vector<Driver> &drivers) {
    vector<double> output(drivers.size(), 0); // initialize
    int orderCount = 0;
    std::priority_queue<Driver> driverPQ(DriverComp);
    std::priority_queue<Order> orderPQ(OrderComp);
    for (int i=0; i<drivers.size(); ++i)
        driverPQ.push(drivers[i]);
    while (orderCount < orders.size()) {
        for (int i=0; i<drivers.size(); ++i) { // push up to m orders.
            orderPQ.push(orders[orderCount])
            orderCount++;
        }
        if (orderCount >= orders.size())
        {break;}
        while (!orderPQ.empty()) {
            Order o = orderPQ.pop();
            Driver d = driverPQ.pop();
            d.rating += o.ratingChange;
            output[d.id] += o.price * o.tipPercent;
            driverPQ.push(d);
        }
    }
    return output;
}
```

This page is intentionally left blank.
You may use this page as working space.