CS2040S — Data Structures and Algorithms
School of Computing
National University of Singapore

Midterm Assessment

9 Mar 2021 **Time allowed:** 2 hours

Instructions — please read carefully:

1. Do not open the midterm until you are directed to do so.
2. Read **all** the instructions first.
3. The quiz is closed book. You may bring one double-sided sheet of A4 paper to the quiz. (You may not bring any magnification equipment!) You may NOT use a calculator, your mobile phone, or any other electronic device.
4. The **QUESTION SET** comprises **TEN (10) questions** and **EIGHTEEN (18) pages**, and the **ANSWER SHEET** comprises of **SIX (6) pages**.
5. The time allowed for solving this test is **2 hours**.
6. The maximum score of this test is **100 marks**. The weight of each question is given in square brackets beside the question number.
7. All questions must be answered correctly for the maximum score to be attained.
8. All questions must be answered in the space provided in the **ANSWER SHEET**; no extra sheets will be accepted as answers.
9. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
10. An excerpt of the question may be provided above the answer box. It is to aid you to answer in the correct box and is not the exact question. You should refer to the original question in the question booklet.
11. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
12. Unless otherwise stated in the question, when we ask for the worst-case big-O running time of an algorithm we always mean to give the tightest possible answer.
13. Unless otherwise stated in the question, we will assume that operators behave as per Java (e.g., 5/2 will evaluate to 2). However, pseudocode does not necessarily satisfy Java syntax (unless stated otherwise) and things that do not compile as legal Java are not necessarily bugs (as long as they are clear pseudocode).
14. Unless otherwise stated in the question, we are not concerned with overflow errors, e.g., storing the value $2^{245,546,983}$ in an integer.

GOOD LUCK!

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: Cheese Jumble [12 marks]

The first column in the table below contains an unsorted list of words. The last column contains a sorted list of words. Each intermediate column contains a partially sorted list.

Each intermediate column was constructed by beginning with the unsorted list at the left and running one of the sorting algorithms that we learned about in class, stopping at some point before it finishes. Each algorithm is executed exactly as described in the lecture notes. One column has been sorted using a sorting algorithm that you have not seen in class. (Recursive algorithms recurse on the left half of the array before the right half.)

Unsorted	**A**	**B**	**C**	**D**	**E**	**F**	**Sorted**
Gouda	Cheddar	Cheddar	Asiago	Asiago	Cheddar	Brie	Asiago
Kanterkaas	Gouda	Gouda	Brie	Brie	Gouda	Asiago	Brie
Cheddar	Jarlsberg	Jarlsberg	Cheddar	Kanterkaas	Jarlsberg	Cheddar	Cheddar
Jarlsberg	Kanterkaas	Kanterkaas	Jarlsberg	Cheddar	Kanterkaas	Derby	Derby
Oaxaca	Oaxaca	Brie	Oaxaca	Jarlsberg	Oaxaca	Edam	Edam
Paneer	Paneer	Feta	Paneer	Manchego	Paneer	Feta	Feta
Manchego	Labneh	Labneh	Manchego	Gouda	Manchego	Gouda	Gouda
Labneh	Manchego	Edam	Labneh	Labneh	Labneh	Labneh	Halloumi
Brie	Brie	Isabirra	Kanterkaas	Feta	Brie	Manchego	Isabirra
Feta	Feta	Derby	Feta	Neufchatel	Feta	Paneer	Jarlsberg
Neufchatel	Neufchatel	Halloumi	Neufchatel	Edam	Neufchatel	Neufchatel	Kanterkaas
Edam	Edam	Asiago	Edam	Isabirra	Edam	Oaxaca	Labneh
Isabirra	Isabirra	Manchego	Isabirra	Derby	Isabirra	Isabirra	Manchego
Derby	Derby	Neufchatel	Derby	Oaxaca	Derby	Jarlsberg	Neufchatel
Halloumi	Halloumi	Oaxaca	Halloumi	Halloumi	Halloumi	Halloumi	Oaxaca
Asiago	Asiago	Paneer	Gouda	Paneer	Asiago	Kanterkaas	Paneer
Unsorted	**A**	**B**	**C**	**D**	**E**	**F**	**Sorted**

Identify, below, which column was (partially) sorted with which of the following algorithms:

1. BubbleSort
2. SelectionSort
3. InsertionSort
4. MergeSort
5. QuickSort (with first element pivot)
6. None of the above.

Hint: Do not just execute each sorting algorithm, step-by-step, until it matches one of the columns. Instead, think about the invariants that are true at every step of the sorting algorithm.

A. What sort was used on Column A? [2 marks]

B. What sort was used on Column B? [2 marks]

C. What sort was used on Column C? [2 marks]

D. What sort was used on Column D? [2 marks]

E. What sort was used on Column E? [2 marks]

F. What sort was used on Column F? [2 marks]

Question 2: Asymptotic Analysis [10 marks]

A. Choose the tightest possible bound for the following function:

$$T(n) = 24n^{0.5}\log^2(n) + 1.7\log^3 n$$

1. $O(1)$
2. $O(\log n)$
3. $O(n)$
4. $O(n \log n)$
5. $O(n \log^2 n)$
6. $O(n^2)$
7. None of the above.

[3 marks]

B. $3^n = O(2^n)$: True or False? [2 marks]

C. $2^{\log_3(n)} = O(n)$: True or False? [2 marks]

D. Suppose that $f(n) = O(g(n))$ and $f(n) = \Omega(h(n))$. Then which of the following statements is/are true?

I It is possible that $h(n) = \Theta(g(n))$.

II It is always the case that $h(n) = O(g(n))$.

III It is always the case that $h(n) = \Omega(g(n))$.

1. Statement I.
2. Statement I and II.
3. Statement I and III.
4. Statement I and II and III.
5. Statement II.
6. Statement II and III.
7. None of the statements are true.

[3 marks]

Question 3: Recurrences [9 marks]

A. Solve the recurrence: $T(n) \leq T(n/c) + 1$, for some constant $c > 1$.

1. $O(1)$
2. $O(\log n)$
3. $O(n)$
4. $O(n \log n)$
5. $O(n^2)$
6. $O(n^3)$
7. $O(2^n)$
8. None of the above.

[3 marks]

B. Which algorithm that we have studied has running time best described by the recurrence: $T(n) \leq T(n/c) + 1$, for some constant $c > 1$.

1. Binary Search
2. MergeSort
3. QuickSelect
4. QuickSort
5. SelectionSort
6. None of the above.

[3 marks]

C. Which recurrence best describes the cost of searching for a key in an (a,b)-tree? (Assume $a > 1$ and $b > 1$ are constant integers with $b > 2a$.)

1. $T(n) \leq T(n/c) + 1$, for some constant $c > 1$.
2. $T(n) \leq T(n/c) + O(n)$, for some constant $c > 1$.
3. $T(n) \leq cT(n/c) + 1$, for some constant $c > 1$.
4. $T(n) \leq cT(n/c) + O(n)$, for some constant $c > 1$.
5. $T(n) \leq T(n/c) + \log n$, for some constant $c > 1$.
6. None of the above.

[3 marks]

Question 4: Algorithm Analysis [9 marks]

For each of the following, choose the best (tightest) asymptotic function from among the given options.

A. What is the asymptotic running time of the following code, as a function of n, when you execute `loopyloop(n, n)`?

```
public static void loopyloop(int a, int b){
    if (b == 1) {
        for (int i = 1; i <= a; i++) {
            System.out.println("Loopy!");
        }
    }
    else loopyloop(a-1, b/2);
}
```

1. $\Theta(1)$
2. $\Theta(\log n)$
3. $\Theta(n)$
4. $\Theta(n \log n)$
5. $\Theta(n^2)$
6. $\Theta(n^3)$
7. $\Theta(2^n)$
8. None of the above.

[3 marks]

B. What is the asymptotic running time of the following code, as a function of n, when you execute `doubleTwist(n)`?

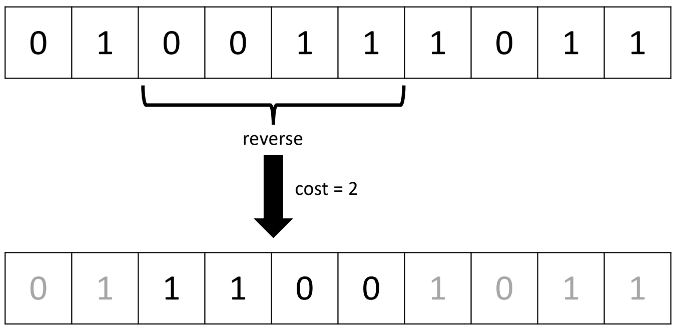
```
public static void doubleTwist(int a){
    if (a == 1) {
        return;
    }
    twist(a/2);
    twist(a/2);
}

public static void twist(int b){
    if (b == 1) {
        return;
    }
    doubleTwist(b/2);
    doubleTwist(b/2);
}
```

1. $\Theta(1)$
2. $\Theta(\log n)$
3. $\Theta(n)$
4. $\Theta(n \log n)$
5. $\Theta(n^2)$
6. $\Theta(n^3)$
7. $\Theta(2^n)$
8. None of the above.

[3 marks]

C. Recall the problem of sorting a binary array by reversing segments of the array. In recitation, we saw a version of the problem where the cost of reversing a segment of the array of size k had cost k. Here we consider a variant of the problem. Assume that the cost of reversing the subarray $A[i, j]$ is equal to the number of 1's in the subarray.



For example, if $A = [0100111011]$ and we reverse the subarray $(2,5)$ which consists of $[0011]$ then the cost is 2.

Consider the divide-and-conquer algorithm presented in recitation:

```
sort(A, begin, end)
    if (end <= begin) return;
    mid = begin + (end-begin)/2;
    sort(A, begin, mid);
    sort(A, mid+1, end);
    reverse(A, begin, end);
```

Assume the initial array A has n elements, n_0 elements equal to 0 and n_1 elements equal to 1. Then what is the cost of the algorithm?

1. $O(n)$
2. $O(n_1)$
3. $O(n \log n_1)$
4. $O(n_1 \log n)$
5. $O(n^2)$
6. $O(n_1^2)$
7. $O(n \log^2 n_1)$
8. $O(n_1 \log^2 n)$

[3 marks]

Question 5: How do they work? [18 marks]

A. Your job is to solve a load balancing problem: you have p servers and T tasks to solve. Your boss suggests using a hash function: for each task t, compute $h(t.name)$ (the hash of the name of the task) and assign the task to the server with that number. (Note there is no hash table here, we are just using the hash function to determine a server.) If you assign tasks in this manner, and if the hash function satisfies the simple uniform hashing assumption, what is the expected number of tasks on each server?

1. p/T
2. T/p
3. T
4. $(p/T)\log p$
5. $(T/p)\log p$
6. $T+p$
7. None of the above.

[3 marks]

B. Which of the following is <u>not</u> true of a properly balanced AVL tree?

1. The height of a leaf is 0.
2. The height of a node is always one less than the height of its parent.
3. The height of a tree is always less than the number of nodes in the tree.
4. If u and v have the same parent, then $|height(u) - height(v)| \leq 1$.
5. If u and v have the same grandparent, then $|height(u) - height(v)| \leq 2$.
6. All of the above are true.

[3 marks]

C. Assume that the following array has just been partitioning used the standard in-place (2-way) partitioning:

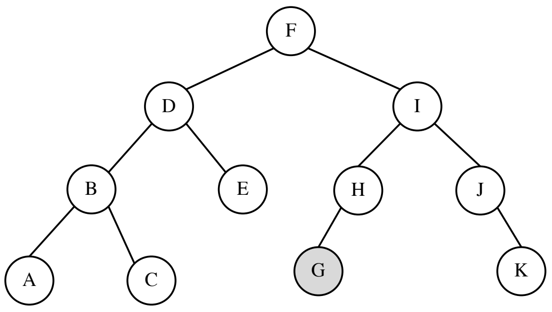
4	17	23	3	5	19	17	4	25	45	29	28

Which of the following values could have been the value of the pivot for this partition operation? (If a value appears more than once, it is a possible pivot if <u>any</u> of the indices containing that value might have been chosen as the pivot.)

1. 5
2. 17
3. 23
4. 25
5. 29
6. None of the above.

[3 marks]

D. Consider the following AVL tree:



The previous operation inserted the node G, which then triggered a double-rotation. Which node was the grandparent of G (i.e., the parent of the parent of G) immediately after G was inserted and before the rotations were performed?

1. D
2. E
3. F
4. H
5. I
6. J.

[3 marks]

E. You have been asked to design an application that can store key/value pairs, supporting insert, delete, and search (by key). For which of the following specifications would you choose a hash table instead of an AVL tree?

I. An application where a slow worst-case operation may cause catastrophic disaster.

II. An application where achieving the maximum average speed is important.

III. An application that can support queries of the form, "find me an entry with a key at least as big as x."

1. Only I.
2. Only II.
3. Only III.
4. I and II.
5. II. and III.
6. None of the above.

[3 marks]

F. Imagine you have a weird hash function h: with probability $1/2$ it maps a key uniformly at random to one of the array slots in the range $[0, m/4 - 1]$; with probability $1/2$, it maps a key uniformly at random to one of the array slots in the range $[m/4, m - 1]$. (Note: this assumption is a replacement for the simple uniform hashing assumption.)

Assume chaining is used to resolve collisions, and that n items have been previously inserted into the hash table. If a new item x is inserted, what is the expected length of the linked list in the bucket $h(x)$ where x is inserted?

1. (n/m)
2. (m/n)
3. $(1/2)(n/m)$
4. $(1/2)(m/n)$
5. $(4/3)(n/m)$
6. $(3/4)(n/m)$
7. $(3/4)(m/n)$
8. None of the above.

[3 marks]

Question 6: Searching [9 marks]

A. Consider the following (pseudocode) implementation of “binary search”:

```
1.      int bSearch(int key, int[] A]
2.          if ((A == null) or (A.length == 0)) return NOT_FOUND;
3.          int begin = 0;
4.          int end = A.length-1;
5.          while (begin != end)
6.              int mid = begin + (end-begin)/2;
7.              if (key <= A[mid]) then
8.                  end = mid-1;
9.              else begin = mid+1;
10.         if (A[begin] == key) return begin;
11.         else return NOT_FOUND;
```

For this problem part, a “bug on line XX” means that <3 characters need to be added/deleted/changed on line XX (with no other changes in the program). The algorithm works correctly if, when A is a sorted array of size at least 1, the algorithm returns the index of the key if it is in the array and NOT_FOUND if the key is not in the array. Which of the following statements is true about the `bSearch` algorithm?

1. The algorithm works correctly.
2. There is a bug in line 5.
3. There is a bug in line 7.
4. There is a bug in line 8
5. There is a bug in line 9
6. The algorithm does not work correctly, but cannot be fixed by altering <3 characters on any one of the specified lines.

[3 marks]

B. Consider the following (pseudocode) implementation of a “special binary search”:

```
int specialSearch(int[] A, int key, int low, int high)
    if ((A == null) or (A.length == 0)) return NOT_FOUND;
    if (low >= high)
        if (A[low] == key) return low;
        else return NOT_FOUND;
    mid = low + (high-low)/2;
    first = specialSearch(A, key, low, mid);
    if (first != NOT_FOUND) return first;
    second = specialSearch(A, key, mid+1, high);
    return second;
```

Which of the following statements is true about the `specialSearch` algorithm?

1. The algorithm has a bug: it may not terminate for certain inputs.
2. The algorithm has a bug: it may return NOT_FOUND, even if the key is in the sorted input array A.
3. The algorithm has a bug: it may return the wrong index, even if the key is in the sorted input array A.
4. The algorithm has a bug: it may return the wrong index, even if the key is <u>not</u> in the sorted input array A.
5. If the algorithm is initially invoked with a sorted array `A` that contains the `key` and `low=0` and `high=A.length-1`, then it satisfies the following invariant (including in every recursive execution): `A[low] <= key <= A[high]`.
6. If the array A is <u>not</u> sorted, the algorithm correctly finds the key and returns its index if it is in the array and returns NOT_FOUND if the key is not in the array.
7. None of the above.

[3 marks]

C. What is the (asymptotic) worst-case running time of the specialSearch algorithm?

1. $\Theta(1)$
2. $\Theta(\log\log(n)$
3. $\Theta(\log(n)$
4. $\Theta(\sqrt{n})$
5. $\Theta(n)$
6. $\Theta(n\log n)$.
7. None of the above.

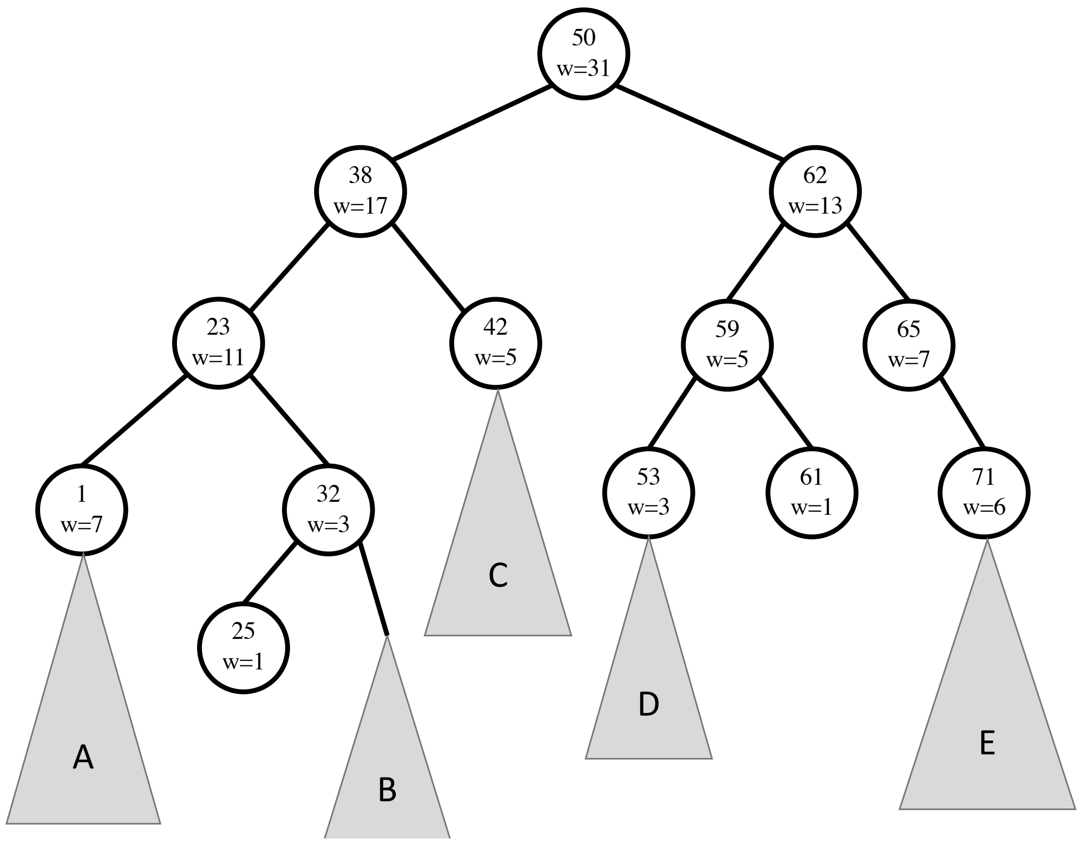
[3 marks]

Question 7: Ice Cream [11 marks]

This year, you have made a resolution to eat more ice cream. As such, you have built a special data structure wherein you can store each day you eat ice cream. The data structure supports two operations:

- `insert(date)`: Adds a day on which you ate ice cream of the specified flavor. We will count days since January 1 (so a date is simply an integer number of days since the year began).
- `check(date1, date2)`: Returns the number of days that you ate ice cream during this period of time (including on the two dates specified).

To implement this, you build a balanced binary tree, augmented with a little bit of extra information. Here is what the tree looks like at the end of March:



Each node in the tree contains the date as a key, along with a weight, which is the number of nodes in the subtree rooted at that node. The triangles (labelled with letters) represent subtrees that are not drawn (but exist in the real tree). You may assume that the tree is height-balanced (even if it is not clear in the picture).

A. Does the tree, where each node contains the date and a weight, contain sufficient augmentation information to implement the `check` query in an asymptotically efficiently manner?

1. Yes: there is enough information to implement the `check` query.

2. No: you must augment the nodes to contain the maximum key in each subtree.
3. No: you must augment the nodes to contain the minimum key in each subtree.
4. No: you must augment the nodes to contain the maximum and the minimum key in each subtree.
5. No: you must augment the nodes to contain some other information unspecified here.

[2 marks]

B. Look at the tree in the figure above, and assume it has whatever is the minimal amount of additional information you think it needs (as per the previous part); do not add unnecessary or redundant information.

The `check` function can be implemented using techniques we have seen in class. Which nodes and subtrees does the execution of `check(32, 59)` traverse or look at? (The list of traversed nodes does not have to be in the order visited, and each node is only listed once even if it is visited more than once, and should include any node that has data that is used by `check`.)

1. 50, 38, 23, 25, 32, B, 42, C, 62, 59, 53, D.
2. 50, 38, 23, 25, 32, 42, C, 62, 59, 53.
3. 50, 38, 23, 25, 32, 42, 62, 59, 53.
4. 50, 38, 23, 25, 32, 62, 59, 53.
5. 50, 38, 23, 32, 62, 59.
6. 32, 53.
7. The entire tree.

[3 marks]

C. What is the value returned by `check(32, 59)` traverse?

1. 6
2. 10
3. 12
4. 13
5. 15
6. 18
7. None of the above.

[3 marks]

D. What is the worst-case asymptotic running time of `check` (implemented efficiently, as in class, where the tree is augmented in the minimal way necessary to implement the query)? Let k be the value returned by `check`.

1. $\Theta(1)$
2. $\Theta(k)$
3. $\Theta(\log n)$
4. $\Theta(\log n + k)$
5. $\Theta(n)$
6. $\Theta(n + k)$
7. None of the above.

[3 marks]

Question 8: Midterm Sorting [16 marks]

Consider the following sorting routines. These sorting algorithms use standard $O(n \log n)$ MergeSort, as defined in class, as a blackbox to sort segments of the array: `MergeSort(int[] A, int low, int high)`. It sorts the array segment `A[low..high]` (inclusive of the end-points) using MergeSort. It also uses a function `isSorted(int[] A)` which returns true when the array is sorted (and false otherwise). The `isSorted` routine takes $\Theta(n)$ time if array A is of size n.

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

Assume we are using `midtermSort(A, k)` to sort an array `A` of unique elements where each item is in an array position at a distance $\leq k$ from its array position in the sorted array. For example, in the following array:

4	5	6	1	2	3	10	11	12	7	8	9

Notice that each element is within distance $k = 3$ of its final position. The value in $A[4] = 2$ belongs in position $A[1]$, and $4 - 1 \leq 3$. Assume $k \geq 2$.

As a matter of notation, when we talk about the prefix of the array $A[0,x]$ we mean the set of array slots $A[0], A[1], \ldots, A[x]$. For example, the prefix of the array $A[0,3]$ in the example above contains $[4,5,6,1]$. More generally, the segment of the array $A[x,y]$ includes all the array elements from $A[x]$ to $A[y]$, inclusive of the endpoints. You may assume that `MergeSort` will work correctly on the elements within the specified range if the range specified is larger than the valid range for A.

A. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ is sorted.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ is sorted.

1. Statement I.
2. Statement II.
3. Both Statements I and II.
4. Neither statement is true.

[2 marks]

B. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ contains the $(j+1)k-1$ smallest elements in the array.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ contains the $(j+2)k-1$ smallest elements in the array.

1. Statement I.
2. Statement II.
3. Both Statements I and II.
4. Neither statement is true.

[2 marks]

C. If an element $A[i]$ is initially in position i and ends in position $i_\ell \le i$ (when the sorting algorithm completes), then at the end of every iteration of the loop it is never moved to a position $i_h > i$: True or False? [2 marks]

D. Assume we run `superMidtermSort(A)` to sort an array `A` of unique elements where each item is in an array position at at distance $\le k$ from its array position in the sorted array. (In this case, note that k is not given to the algorithm.) What is the running time of the algorithm as a function of n and k? Give the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$
4. $\Theta(k \log n)$
5. $\Theta(n \log^2 k)$
6. $\Theta(n^2)$
7. $\Theta(nk)$.
8. None of the above.

E. Assume we run `superMidtermSort(A)` to sort an array `A` with the possibility of repeated elements. Is the resulting sorting algorithm stable? [2 marks]

F. Is the `superMidtermSort(A)` an in-place sorting algorithm? [2 marks]

G. Assume instead we run `InsertionSort(A)` to sort the array `A` of unique elements where each item is in an array position at at distance $\leq k$ from its array position in the sorted array. What is the running time of the algorithm as a function of n and k? Give the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$
4. $\Theta(k \log n)$
5. $\Theta(n \log^2 k)$
6. $\Theta(n^2)$
7. $\Theta(nk)$.
8. None of the above.

Question 9: Scheduling [6 marks]

You are building a job scheduler. Each job has a unique name, a beginning time t_1, a deadline t_2, and a cost c. You are supposed to build a data structure that supports the following operations:

- `insert(jobname, begin, end, cost)` : adds the job to the data structure.
- `delete(job-name)` : removes the job from the data structure.
- `count(time)` : returns the number of jobs that have a deadline prior to the specified time.

A. Which data structure that we have studied in class solves this problem most efficiently (with no significant modification)? (Our goal is for all three operations to be efficient.)

1. Array, sorted by completion time.
2. AVL tree (supporting typical dictionary operations)
3. Interval tree (supporting insertion and deletion of intervals, and point queries)
4. Order statistics tree (supporting insertion, deletion, rank, and select operations)
5. Hash table (supporting typical symbol table operations)
6. None of the above are sufficient

[2 marks]

B. Recall that each job has a cost. The cost may be positive or negative (depending on whether the payment for completing the job is more or less than the cost of the resources needed to complete the job). Assume we decide to store the jobs in an array, sorted by deadline. (Note that this choice is unrelated to the correct answer to the previous part, and may or may not be a good choice.)

For a given budget B, we want to find a time t where the total cost of all the jobs with deadline $<= t$ is $< B$ (since our goal is to not spend more than B in completing the jobs). For this part of the problem, assume the set of jobs is fixed, i.e., we are not worried about inserting or deleting jobs.

In each cell in the array, we store some additional information to help find this. Assume the array storing the jobs is J.

- We store in cell $J[i]$ the minimum cost of a job in the prefix `J[0..i]`. We call this the "prefix min."
- We store in cell $J[i]$ the maximum cost of a job in the prefix `J[0..i]`. We call this the "prefix max."
- We store in cell $J[i]$ the sum of the costs of the jobs in the prefix `J[0..i]`. We call this cost the "prefix sum."
- We store in cell $J[i]$ the minimum prefix sum of a job in the prefix `J[0..i]`. We call this the "prefix sum min."
- We store in cell $J[i]$ the maximum prefix sum of a job in the prefix `J[0..i]`. We call this the "prefix sum max."

As an example, consider the following set of jobs (with their deadline and cost specified), with the various additional data filled in:

Deadlines	5	7	9	11	20	30	32
Costs	-1	1	3	5	1	1	1
prefix min	-1	-1	-1	-1	-1	-1	-1
prefix max	-1	1	3	5	5	5	5
prefix sum	-1	0	3	8	9	10	11
prefix sum min	-1	-1	-1	-1	-1	-1	-1
prefix sum max	-1	0	3	8	9	10	11

Which of the following statements is true:

1. We can use binary search on the prefix mins to find a prefix with cost $< B$.
2. We can use binary search on the prefix sums to find a prefix with cost $< B$.
3. We can use binary search on the prefix sum mins to find a prefix with cost $< B$.
4. We can use binary search on the prefix sum maxs to find a prefix with cost $< B$.
5. We cannot use binary search on any of the values defined (i.e., the prefix min, prefix max, prefix sums, prefix sum mins, or prefix sum maxs) to solve this problem.