Assignment 2

CS-GY 6033 INET Spring 2025

Due date: 11:55pm on **March 1st 2025 on Gradescope**.

Instructions:
Below you will find the questions which make up your homework. They are to be written out (or typed!) and handed in online via Gradescope before the deadline.

Question 1: Hashing

.

(a) 5 points Consider a hash table implemented as an array, $T[]$, indexed from $0 \ldots 12$. Hashing is carried out using double hashing, with $h_1(k) = k^2 \mod 13$ and $h_2(k) = k^2 + 1 \mod 13$. Show the insertion of: $45, 10, 100, 17, 4, 26, 24, 6, 9$, or explain why it is that certain keys are not inserted.
Explain whether or not the final table *could* look different, if the items were inserted in a different order.

(b) 5 points Repeat the above, but where hashing is carried out using the quadratic hash function $h(k, i) = k + 2i + 3i^2 \mod 13$. Show the insertion of the keys from part (a), or explain why it is that certain keys are not inserted.

(c) 3 points Repeat the hashing once again, this time using function $h(k) = k^2 \mod 13$, where collisions are resolved with chaining. Note that each *insert* can be done at the *end* of the chain in constant time, by storing a reference to the end of the linked list.

(d) 3 points Suppose we wish to search for the key 9 in the above hash tables. How many probes are required to find key 6 for each table?

(e) 4 points Consider a hash table $T[0, 1, 2, 3, 4, 5, 6]$ which uses quadratic hashing given by: $h(k, i) = k + 2i + 2i^2 \mod 7$. Suppose the table is currently populated as $T[0] = 7, T[3] = 3, T[4] = 4, T[5] = 5$ and the remaining spots are empty. Give an example of a key that **fails to insert** into the table, under the above hash function. Explain why this is possible, even though there are free spots in the table. What collision technique **guarantees** to find a free spot in the table, whenever one exists?

Question 2: Selection

.

(a) 6 points Carry out the *Select* algorithm on set below using $k = 14$ (return the element of rank 14). Show your steps! You do not need to show the steps of the partition algorithm. Instead, ensure that the output of the partition algorithm maintains the elements in their original relative order. You must carry out *all* recursive calls to the Select algorithm. You may assume that the base case of the Select algorithm is any input of at most 5 elements, in which case it returns the element of rank k.
**When taking the median of an even number of elements, use the lower median by default*

$$25, 37, 9, 52, 14, 7, 19, 89, 35, 42, 83, 15, 4, 53, 31, 40, 86, 99, 46, 66, 34, 5, 22, 2, 8, 90, 10, 18, 30, 68,$$

$$21, 17, 84, 20, 29, 77, 12, 16, 45, 33, 6, 41, 19, 53, 42, 93, 3, 23, 18, 91, 87, 1, 11, 88, 86$$

(b) 8 points Suppose we alter the Select(k) algorithm from class, so that it groups the elements into groups of size 4 instead of groups of size 5. The main steps of the algorithm are:

1. Sort each group of size 4. If there is only one group, return the element of rank k.
2. Identify the 2nd smallest element of each group (the lower median). Make a recursive call to find the median of the medians. Call this element x
3. Partition the input into those that are smaller and those that are larger than x. Let r be the rank of element x.
4. If $k = r$,return x
5. If $k < r$, recurse on the set of smaller elements. Otherwise recuse on the larger elements.

Your job: Determine the runtime of each of the above steps. Write the recurrence relation for this algorithm. Show that the overall runtime of this version of Select is $O(n)$ using the substitution method.

Question 3: Heaps

.

(a) 4 points
Consider the following algorithm below for building a max-heap on n elements. The initial call is MakeHeap$(A, 1)$.

```
Make-Heap($A, i$)
    if $i \leq$ A.heapsize
        Bubble-down($A, i$)
        Make-Heap($A, i + 1$)
```

Does this algorithm correctly build a max-heap? Justify your answer.

(b) 4 points Consider a max-heap that contains the elements $1, 2, 3, 4, 5$. Draw all possible heaps on these five elements.

(c) 6 points Suppose your application requires a data structure that carries out inserts, and remove max operations. Answer each of the following:

- Suppose there are many insert operations, operations, but only a few remove the maximum operations. Which implementation do you think would be most effective: heap, unordered array, or ordered array? Ensure that you justify your answer. Note that in the unordered array option, the implementation keeps track of the current maximum element.
- Repeat the above assuming there are many find-max operations, but a small number of insert and remove-max operations.

(d) 8 points A *min-max-heap* is a complete binary tree containing alternating *min* and *max* levels. The root node is the **minimum** of all nodes in the tree. The nodes at the next level are the largest nodes of their subtrees. For each level after that, a node that is on a min level is the minimum of all nodes in its subtree, and a node that is on a max level, is the maximum of all nodes in its subtree. Below is an example of a min-max tree. Assume that the elements are stored in array $A[]$ as for usual heaps, where $A.heapsize$ indicates the size of the heap. Your job is fill in the skeleton pseudo-code below, so that it correctly o **inserts** into this type of heap. Call your procedure MinMaxInsert(A, k), which inserts k into the heap, and carries out the necessary swaps in order to maintain the min-max heap property. The procedue references MinBubbleUp(A, i) and MaxBubbleUp(A, i) which carry out the bubble up procedure assuming i is either at a min level or max level.

```
MinMaxInsert($A, k$)
    A.heapsize++
    ____________ $= k$
    $L =$ ____________ # Level of the new node
    if $L$ is even:
        if $A$[          ] $< A[i]$
            Swap ____________
            __________________
        else
            __________________
    else
        if $A$[          ] $> A[i]$
            Swap ____________
            __________________
        else
            __________________
```

```
MinBubbleUp($A, i$)
    if __________________
        parent = ____________
        grand = ____________
        if $A[i] <$ ____________
            __________________
            __________________
```

```
MaxBubbleUp($A, i$)
fill in
```

Question 4: Lower Bounds and Linear time Sorting

.

(a) 6 points

- Let $A[]$ be an array of n natural numbers, written in binary, where each number is at most 4^n.
- Let $B[]$ be an array of n integers, where each integer is in the range $-n^2, \ldots, n^2$.

For each input array A and B, describe how to sort the input using radix sort. Justify the runtime.

(b) 4 points Draw the decision tree for binary search on the elements A, B, C, D, E, F, G where the search is for key k. Recall that the input to binary search is sorted in increasing order. The decision tree in this case is such that each node has **three children:** one for $=$, one for $<$, and one for $>$. Note that the nodes of the tree will be comparisons between k and one of the input letters.

(c) 4 points Explain why bucket sort does not necessarily have an expencted runtime of $O(n)$ if the input is not uniformly distributed.

(d) 4 points Explain how to use bucket sort when the input is a set of uniform random **integers** in the range 0 to 100. Justify that the expected runtime is $O(n)$.