CS61BL Final Review Solution

1 Heaps of Fun

(a) Consider the following array representation of a min-heap.

```
   i   0  1  2  3  4  5  6  7  8  9  10  11
a[i]   *  A  C  P  M  G  Q  X  Z  T   L   O
```

Show the array after performing each of the following operations on the *original* min-heap.

1. Insert E.

a[i]	*	A	C	E	M	G	P	X	Z	T	L	O	Q

2. Delete the minimum.

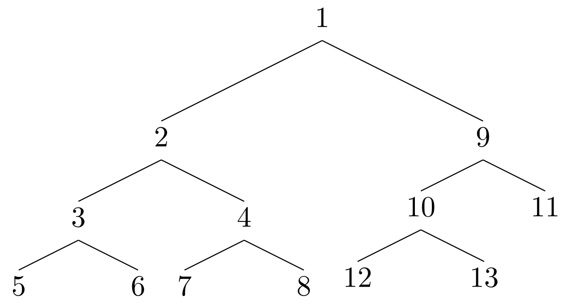
a[i]	*	C	G	P	M	L	Q	X	Z	T	O		

(b) Which of the following are advantages of a heap (with a resizing array) over a sorted linked list?

expected time for `insert` is lower **T**	`insert` has lower worst-case runtime **F**
expected time for `delMin` is lower **F**	`delMin` has lower worst-case runtime **F**
expected storage cost is lower **T**	`max` has lower worst-case runtime **F**

(c) What is the size of the largest binary min-heap in which the fifth largest element is a child of the root? *Assume the heap has no duplicate elements.*

13 elements.



2 SixtyOnePQ

You have been hired by Alan to help design a priority queue-like data structure for Kelp ordered on the timestamp of each `Review`. This data structure needs to support the following operations:

- `insert(Review r)` a `Review` in $O(\log N)$ time.
- `edit(int id, String body)` any one `Review` in $\Theta(1)$ time. (Timestamp does not change.)
- `sixtyOne()` — return the sixty-first latest `Review` in $\Theta(1)$ time.
- `pollSixtyOne()` — remove and return the sixty-first latest `Review` in $O(\log N)$ time.

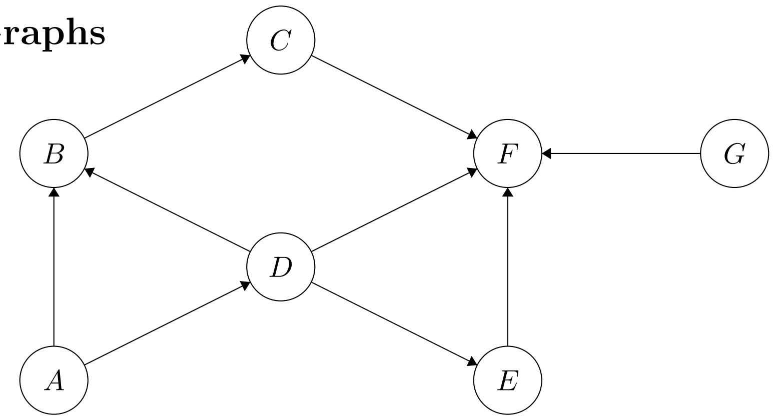
Explain how you would implement the required functionality, using one or more data structures that we have seen in class. Write pseudocode for each of the operations listed above. You may assume that $N > 61$ and omit all checks for smaller N.

Your implementation should support finding the k^{th} smallest item with an asymptotic runtime independent of k. That is, it should be possible to change 61 to another constant while maintaining the asymptotic runtime bound. For instance, we should be able to find the median by setting $k = \frac{N}{2}$.

Maintain a max-heap called `firstSixtyOne` with 61 `Reviews`, a min-heap called `olderReviews` with all the rest, and a `HashMap` mapping any given integer ID to its corresponding `Review`.

```java
class SixtyOnePQ {

    PriorityQueue<Review> firstSixtyOne;
    PriorityQueue<Review> olderReviews;
    HashMap<Integer, Review> idToReview;

    void insert(Review r) {
        firstSixtyOne.offer(r);
        idToReview.put(r.id, r);
        if (firstSixtyOne.size() > 61)
            olderReviews.offer(firstSixtyOne.poll());
    }

    void edit(int id, String body) {
        idToReview.get(id).body = body;
    }

    Review sixtyOne() {
        return firstSixtyOne.peek();
    }

    Review pollSixtyOne() {
        Review r = firstSixtyOne.poll();
        idToReview.remove(r.id);
        if (!olderReviews.isEmpty())
            firstSixtyOne.offer(olderReviews.poll());
        return r;
    }
}
```

3 Graph-Graphs



(a) Show the adjacency matrix and adjacency list representations of the above graph.

	A	B	C	D	E	F	G
A	0	1	0	1	0	0	0
B	0	0	1	0	0	0	0
C	0	0	0	0	0	1	0
D	0	1	0	0	1	1	0
E	0	0	0	0	0	1	0
F	0	0	0	0	0	0	0
G	0	0	0	0	0	1	0

A	B, D
B	C
C	F
D	B, E, F
E	F
F	
G	F

(b) With an **adjacency list**, how long does it take to check for the existence of an edge between two vertices? Give the tightest possible asymptotic bound in terms of $|V|$ and $|E|$.

$O(|V|)$

(c) How long does the same operation take with an **adjacency matrix**?

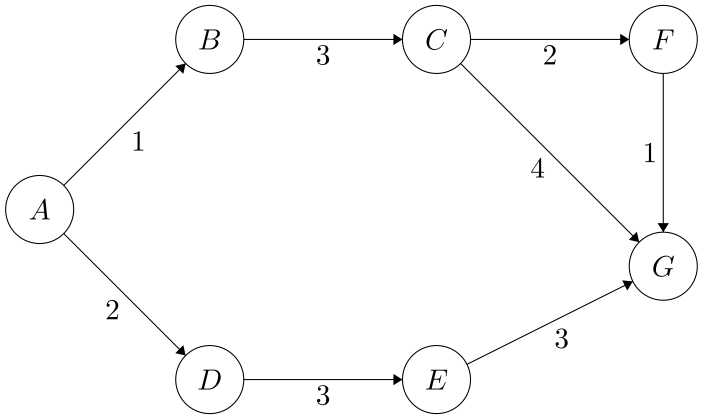
$\Theta(1)$

(d) Give the DFS preorder, DFS postorder, and BFS traversals starting from vertex A as well as the DFS postorder topological sort. Break ties alphabetically.

`DFS preorder`	A	B	C	F	D	E	
`DFS postorder`	F	C	B	E	D	A	
`BFS`	A	B	D	C	E	F	
`Topological`	G	A	D	E	B	C	F

4 Circle the Mascot

$edge(A,B) = 1$	$h(A,G) = 8$
$edge(B,C) = 3$	$h(B,G) = 6$
$edge(C,F) = 2$	$h(C,G) = 5$
$edge(C,G) = 4$	$h(F,G) = 1$
$edge(F,G) = 1$	$h(D,G) = 6$
$edge(A,D) = 2$	$h(E,G) = 3$
$edge(D,E) = 3$	
$edge(E,G) = 3$	



(a) Run Dijkstra's Algorithm to find the shortest paths from A to every other vertex.

$A \to B = 1$ $\quad A \to C = 4$ $\quad A \to D = 2$

$A \to E = 5$ $\quad A \to F = 6$ $\quad A \to G = 7$

(b) Give the path A* Search would return given the heuristic above and A as the start and G as the goal.

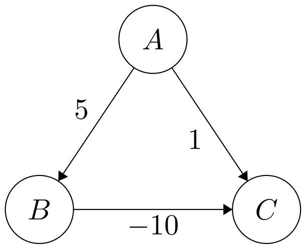
$$A \to D \to E \to G$$

(c) The above heuristic is *inadmissible*: the heuristic returns a greater length than the actual remaining distance needed to reach the goal. Determine the possible range for each inadmissible heuristic value so that the heuristic becomes admissible.

$0 \le h(A,G) \le 7$ $\qquad 0 \le h(C,G) \le 3$

(d) Is it true that, assuming every vertex is reachable from a given source, Dijkstra's Algorithm always finds a shortest path from that source to every vertex?

False. Dijkstra's assumes that, once we visit a vertex, the algorithm has found the shortest path to that vertex. Because Dijkstra's checks against the closed set of visited vertices before processing a new vertex, the optimal path will not overwrite the existing shortest path. In the example graph, we will explore $A \to C$ first but fail to replace it with the optimal shortest path $A \to B \to C$.



(e) Given an undirected, positively-weighted graph $G = (V, E)$, a set of start vertices S, and a set of end vertices T, describe an efficient algorithm that returns the shortest path starting from any one vertex in S and ending at any one vertex in T.

Add a vertex, S^*, connected to all start vertices and a vertex, T^*, connected to all end vertices with any non-negative weight k. Using Dijkstra's Algorithm, find the shortest path from S^* to T^* and then remove S^* and T^* from the final result.

5 Sort This Out

For each of the following scenarios, choose the best sorting algorithm and explain your reasoning.

(a) Sorting a list created by first taking a sorted list of N elements and then swapping N pairs of adjacent elements.

Insertion sort, since a list created in such a manner will have at most N inversions. (Recall that insertion sort runs in $\Theta(N + K)$ time, where K is the number of inversions.) Bubble sort is also correct since that would also take $\Theta(N)$ time on such a list.

(b) Sorting a list of everyone who took the US census by last name.

MSD radix sort. LSD radix sort would work, but we would have to pad each last name with extra characters so that all names were the same length.

(c) Sorting a list on a machine where swapping two elements is much more costly than comparing two elements (and you want to do the sort in place).

Selection sort, since in its most common implementation, selection sort performs N swaps in the worst case, whereas all other common sorts perform $\Omega(N \log N)$ swaps in at least some cases.

(d) Sorting a list so large that not all of the data will fit into memory at once. As is, at any given time most of the list must be stored in external memory (on disk), where accessing it is extremely slow.

Merge sort is ideal here since its divide-and-conquer strategy works well with the restriction on only being able to hold a partition of the list in memory at any given time. Sorted runs of the list can be merged in memory and committed to disk storage one block at a time, minimizing disk reads and writes.

Each of the following sequences represent an array being sorted at some intermediate step. Match each sample with one of the following sorting algorithms: **insertion sort, selection sort, heap sort, merge sort, quicksort, LSD radix sort, MSD radix sort**. The original array is below.

	`5103 9914 0608 3715 6035 2261 9797 7188 1163 4411`	
(a)	`5103 9914 0608 3715 2261 6035 7188 9797 1163 4411`	Merge sort
	`0608 2261 3715 5103 6035 7188 9797 9914 1163 4411`	
(b)	`0608 1163 5103 3715 6035 2261 9797 7188 9914 4411`	Selection sort
	`0608 1163 2261 3715 6035 5103 9797 7188 9914 4411`	
(c)	`9797 7188 5103 4411 6035 2261 0608 3715 1163 9914`	Heap sort
	`4411 3715 2261 0608 1163 5103 6035 7188 9797 9914`	
(d)	`2261 4411 5103 1163 9914 3715 6035 9797 0608 7188`	LSD radix sort
	`6035 5103 1163 7188 2261 4411 0608 3715 9797 9914`	
(e)	`5103 0608 3715 2261 1163 4411 6035 9914 9797 7188`	Quicksort
	`0608 2261 1163 3715 5103 4411 6035 9914 9797 7188`	
(f)	`0608 5103 9914 3715 6035 2261 9797 7188 1163 4411`	Insertion sort
	`0608 2261 3715 5103 6035 9914 9797 7188 1163 4411`	

6 Disjoint Sets & Kruskal's Algorithm & ... Regex

(a) Give the tightest asymptotic bound on the height of a weighted quick union tree in terms of N, the number of elements. Provide a $\Theta(\cdot)$ bound if possible, else give both $O(\cdot)$ and $\Omega(\cdot)$.

$\Omega(1), O(\log N)$

(b) Is it true that, given a graph G, if we add some constant k to every edge weight, G's minimal spanning tree(s) remain unchanged?

True. If $|V|$ is the number of vertices, the MST must have $|V|-1$ edges. Therefore, since the total weight of any tree with $|V|-1$ edges would increase by $k|V|-k$, any minimal tree will remain minimal relative to all other trees.

(c) Assume we are using disjoint sets with path compression. How many calls to `find` need to be made in order for each node to be directly connected to the root node?

We need to call `find` as many times as the number of leaves in the disjoint set.

(d) In terms of runtime, what is the worst way to place the integers 1, 2, 3, 4, 5 into the same disjoint set? Give an answer in the form of a series of calls to the `connect` method.

`connect(1,2), connect(2,3), connect(3,4), connect(4,5)`

(e) What are the shortest and tallest heights possible for a Quick Union with N elements? What does this mean for the best and worst-case runtime for `isConnected` and `connect`?

Shortest: 1, Tallest: $N-1$
`isConnected` best case: $\Theta(1)$, worst case $\Theta(N)$
`connect` best case: $\Theta(1)$, worst case $\Theta(N)$

(f) What is the shortest and tallest height possible for a Weighted Quick Union with N elements? What does this mean for the best and worst-case runtime for `isConnected` and `connect`?

Shortest: 1, Tallest: $\log_2 N$
`isConnected` best case $\Theta(1)$, worst case $\Theta(\log N)$
`connect` best case $\Theta(1)$, worst case $\Theta(\log N)$

(g) Mark all of the strings matched by the regular expression `((C|S)* 61*BL+)`.

`CSS 6BL     S 61BL     CS 61BL`

(h) Write a regular expression that matches any binary string of length 8, 16, 32, or 64.

`^(0|1){64}|(0|1){32}|(0|1){16}|(0|1){8}$`

(i) A certain type of bracketed list consists of words composed of one or more lower-case letters alternating with unsigned decimal numerals and separated by commas, as in these examples:

`[]     [wolf, 12, badger, 11]     [dog,10, cat]`

Commas may be followed (but not preceded) by blanks, and the list may have odd length (ending in a word with no following numeral). There are no spaces around the `[]` brackets. Write a regular expression that matches such lists.

`\[(([a-z]+, *\d+, *)*[a-z]+(, *[0-9]+)?)?\]`