

1. Binary Heap

- a. Show the result of enqueueing 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2 into a minimum binary heap. You may show the heap as a tree structure (even though our implementation stored the heap in an array).

Show the resulting binary heap from above after doing 3 dequeue operations.

- b. Given the list: [10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, 2], Show the minimum heap resulting from a “bottom-up” construction. You may show the heap as a tree structure (even though our implementation stored the heap in an array).

8. Hash Tables

a. Given the input:

255 375 254 639 718 101 100 692 538 800 277

a fixed table size of 10, and a hash function of $H(x) = x \bmod 10$, show the resulting separate chaining hash table. Why is 10 a bad choice for a table size?

What is the load factor of this table? What is the maximum recommended load factor for a table using separate chaining? What may happen if this load factor is exceeded?

a. Given the input:

375 254 639 718 101 105 692 538

a fixed table size of 10, and a hash function of $H(x) = x \bmod 10$, show the resulting linear probing hash table. What is the load factor of this table? What is the maximum recommended load factor for a table using linear probing?

b. Given the input:

255 375 254 639 718 101 100 692 538 800 277

a fixed table size of 10, and a hash function of $H(x) = x \bmod 10$, show the resulting quadratic probing hash table.

Indicate if any inputs can't fit in the table. What is the load factor of this table? What is the maximum recommended load factor for a table using quadratic probing? What may happen if this load factor is exceeded?

What are the load factor limitations of the following?

- a) Separate Chaining?
- b) Linear Probing?
- c) Quadratic Probing?

Sorting

```
// index 0   1   2   3   4   5   6   7   8
        { 16, 21, 45, 8, 11, 53, 3, 26, 49}
```

9. Trace the execution of the **selection** sort algorithm over the array above. Show each pass of the algorithm and the state of the array after the pass has been performed, until the list is sorted.

```
// index 0   1   2   3   4   5   6   7   8
        { 16, 21, 45, 8, 11, 53, 3, 26, 49}
```

10. Trace the execution of the **insertion** sort algorithm over the array above. Show each pass of the algorithm and the state of the array after the pass has been performed, until the list is sorted.

11. What are the best, average, and worst Big-Oh descriptions of selection sort and insertion sort?

12. Merge Sort: Finish the merge sort code. Assume that you can call a merge function that takes 3 lists and will merge 2 of the lists into the other list.

```
def merge_sort(list1):
    if len(list1) > 1:
        mid = len(list1) // 2
        left_half = list1[:mid]
```