IEEE CP SMP 2018
Assignment 2
Topic: Time Complexity

Suraj Singh
1800-163-24

1. **Stacks**
   - top(): O(1)
   - push(): O(1)
   - pop(): O(1)
   - size():O(1)

2. **Queues**
   - front() : O(1)
   - back(): O(1)
   - push(): O(1)
   - pop(): O(1)
   - size(): O(1)

3. **Vectors**
   - push_back() : O(1)
   - size(): O(1)
   - find(): O(1)
   - erase(): O(n)
   - sort() : O(n*log(n))
   - iterating through the vector - O(n)

4. **Arrays**
   - inserting at a position - O(1)
   - sort() - O(n*log(n))
   - lower_bound() - O(log(n))
   - upper_bound() - O(log(n))
   - next_permutation() - O(n)
   - prev_permutation() - O(n)

5. **Pair**
   - inserting at a position - O(1)
   - sort() - O(nlog(n))

- printing the array - O(n)

6. **Priority Queue**

(They're implemented using heaps in STL, that's why log(n))
- push() - O(log(n))
- top() - O(1)
- empty() - O(1)
- pop() - O(log(n))

7. **Map**

(implemented using red black trees, hence log(n) - the height of the tree)
- insertion() - O(log(n))
- find() - O(log(n))
- iterating through all elements - O(n)

8. **Set**

(implemented using red black trees, hence log(n) - the height of the tree)
- insert() - O(log(n))
- size() - O(1)
- erase() - O(1) + balancing the tree would take O(n)
- find() - O(log(n))

9. **MultiSet**

(implemented using red black trees, hence log(n) - the height of the tree)
- insert() - O(log(n))
- erase() - O(1) + balancing the tree would take O(n)
- find() - O(log(n))

10. **Double Ended Queue**

(implemented as a vector so accessing and adding elements at the front and rear should take ammortized constant time)
- front() - O(1)
- back() - O(1)
- push_front() - O(1)
- push_back() - O(1)
- pop_front() - O(1)
- pop_back() - O(1)