CS2040S Tutorial 2

AY 24/25 Sem 2 — github/omgeta

Q1. (a.) $T(n) = O(n)$

(b.) $T(n) = T(\frac{n}{2}) + O(n) = O(n)$

(c.) $T(n) = O(n^2)$

(d.) $T(n) = 2T(\frac{n}{2}) + n = O(n \log n)$

(e.) $T(n) = O(\phi^n)$

(f.) $T(n) = \log(1) + \log(2) + \cdots + \log(n) < \log(n) + \log(n) + \cdots + \log(n) = O(n \log n)$

(g.) $T(n) = O(n^2)$

Q2. (a.) $T(n) = T(n-1) + O(n) \leq c1 + c2 \cdots + cn = \frac{cn(n+1)}{2} = O(n^2)$

```
private static void sort(int[] A, int n) {
    if (n > 0) {
        sort(A, n-1);
        int x = A[n];
        int j = n-1;
        while (j >= 0 && A[j] > x) {
            A[j+1] = A[j];
            j--;
        }
        A[j+1] = x;
    }
}
```

(b.)
1. First, use SelectionSort to sort by b.
2. Second, use MergeSort to sort by a.
3. Since MergeSort is stable, it preserves the order of the pairs being sorted by b in 1.

(c.) $T(n) = O(n \log n)$, $S(n) = O(n)$

```
private static void sort(int[] a) {
  if (a.length <= 1) return;
  int[] tmp = new int[a.length];

  int n = 1;
  while (n < a.length) {
      for (int left = 0; left < a.length; left += 2 * n) {
          int mid = Math.min(left + n, a.length);
          int right = Math.min(left + 2 * n, a.length);

          merge(a, tmp, left, mid, right);
      }

      for (int i = 0; i < a.length; i++) a[i] = tmp[i];

      n *= 2;
  }
}

private static void merge(int[] src, int[] dst, int left, int
    mid, int right) {
  int i = left, j = mid, k = left;

  while (i < mid && j < right)
      if (src[i] <= src[j]) dst[k++] = src[i++];
      else dst[k++] = src[j++];

  while (i < mid) dst[k++] = src[i++];
  while (j < right) dst[k++] = src[j++];
}
```

Q3. (a.) Use a pointer to keep track of where the head/tail of the stack/queue is

(b.) Keep a pointer of both the head and tail of the deque.

(c.) We would need to throw an error or fail silently if attempts are made to remove from an empty ADT, or add to a full ADT

(d.) Solution:

```
private static boolean check(String s) {
  Stack stack = new Stack();
  for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    if (c == '(') stack.push(c);
    else if (c == ')')
      if (stack.isEmpty() || stack.pop() != '(') return false;
  }
  return stack.isEmpty();
}
```

(e.) Solution:

```
private static boolean check(String s) {
  Stack stack = new Stack();
  for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    if (c == '(' || c == '{' || c == '[') stack.push(c);
    else if (c == ')' || c == '}' || c == ']')
      if (stack.isEmpty() || stack.pop() == getOpening(c))
   return false;
  }
  return stack.isEmpty();
}

private static char getOpening(char c) {
  if (c == ')') return '(';
  else if (c == '}') return '{';
  else if (c == ']') return '[';
  else return null;
}
```

Q4. Solution: Traverse the elements, adding them to the stack. If the current element $\leq$ previous element, remove them from the stack. If the stack is empty, then the current visibility is infinite, else print the number of elements in the stack.

Q5. Perform an iterative MergeSort in the same way as before but using the two queues.