CS 61B Fall 2024

ADTs, Asymptotics II, BSTs

Discussion 06: October 7, 2024

1 ADT Matchmaking

Match each task to the best Abstract Data Type for the job and justify your answer (ie. explain why other options would be less ideal). The options are ~~`List`~~, `Map`, ~~`Queue`~~, `Set`, and ~~`Stack`~~. Each ADT will be used once.

1. You want to keep track of all the unique users who have logged on to your system.

   Set, as we track uniqueness, reps not allowed

2. You are creating a version control system and want to associate each file name with a Blob.

   Map, as we need key value association

3. We are grading a pile of exams and want to grade starting from the top of the pile (*Hint: what order do we pile papers in?*).

   Stack, as we are using LIFO

4. We are running a server and want to service clients in the order they arrive.

   Queue, as we need order and serve them in their order arrival

5. We have a lot of books at our library and we want our website to display them in some sorted order. We have multiple copies of some books and we want each listing to be separate.

   List, as we need order and reps are allowed

Some geometric sums you may find helpful in the rest of the worksheet:

$$1 + 2 + 3 + 4 + 5 + \cdots + N \in \Theta(N^2)$$
$$1 + 2 + 4 + 8 + 16 + \cdots + N \in \Theta(N)$$

General case:

$$1 + 2 + 3 + 4 + 5 + \cdots + f(N) \in \Theta(f(N)^2)$$
$$1 + 2 + 4 + 8 + 16 + \cdots + f(N) \in \Theta(f(N))$$

2 I Am Speed

(a) For each code block below, fill in the blank(s) so that the function has the desired runtime. Do not use any commas. If the answer is impossible, just write "impossible" in the blank. Assume that `System.out.println` runs in constant time. You may use Java's `Math.pow(x, y)` to raise `x` to the power of `y`.

```
// Desired Runtime: Θ(N)
public static void f1(int N) {
    for (int i = 1; i < N; _i++_____){
        System.out.println("hi Dom");
    }
}


// Desired Runtime: Θ(log N)
public static void f2(int N) {
    for (int i = 1; i < N; _i*=2____) {
        System.out.println("howdy Ergun");
    }
}


// Desired Runtime: Θ(1)
public static void f3(int N) {
    for (int i = 1; __i<C_____; i += 1) {
        System.out.println("hello Anniyat");
    }
}


// Desired Runtime: Θ(2^N)
// This one is tricky! Hint: think about the dominating term in 1 + 2 + 4 + 8 + ... + f(N)
public static void f4(int N) {
    for (int i = 1; Math.pow(2,N)__; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("what's up Alyssa");
        }
    }
}
```

some constant indep of N

(b) *Extra* Give the worst case and best case running time in $\Theta(\cdot)$ notation in terms of M and N. Assume that `kachow()` runs in $\Theta(N^2)$ time and returns a `boolean`.

```
1  for (int i = 0; i < N; i += 1) {
2      for (int j = 1; j <= M; ) {
3          if (kachow()) {
4              j += 1;
5          } else {
6              j *= 2;
7          }
8      }
9  }
10
```

Best: $N^3 \log M$

worst: $N^3 M$

worst one is when kachow is always true, so we get N^2 of kachow, times N of outer loop and times M of inner loop

with Best case, kachow is always false, but we still have to check it every time, so we get N^2 of kachow times N of outer loop, and times $\log M$ of inner loop as we do else statement

3 Re-cursed with Asymptotics!

(a) What is the runtime of the code below in terms of n?

```
1  public static int curse(int n) {
2      if (n <= 0) {
3          return 0;
4      } else {
5          return n + curse(n - 1);
6      }
7  }
8
```

Best: $\Theta(1)$ as $n \leq 0$

if we've n=3, then 3 + curse(2)

3 + 2 + curse(1)

3 + 2 + 1 + curse(0)

3 + 2 + 1 + 0

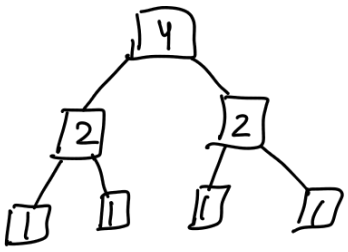
so worst case will be, $\Theta(N+1) = \Theta(N)$

can be represented as tree as well ↓

(b) Can you find a runtime bound for the code below? We can assume the System.arraycopy method takes $\Theta(N)$ time, where N is the number of elements copied. The official signature is System.arrayCopy(Object sourceArr, int srcPos, Object dest, int destPos, int length). Here, srcPos and destPos are the starting points in the source and destination arrays to start copying and pasting in, respectively, and length is the number of elements copied.

```
1  public static void silly(int[] arr) {
2      if (arr.length <= 1) {
3          System.out.println("You won!");
4          return;
5      }
6
7      int newLen = arr.length / 2;
8      int[] firstHalf = new int[newLen];
9      int[] secondHalf = new int[newLen];
10
11     System.arraycopy(arr, 0, firstHalf, 0, newLen);
12     System.arraycopy(arr, newLen, secondHalf, 0, newLen);
13
14     silly(firstHalf);
15     silly(secondHalf);
16 }
17
18
```

if arr.len = 4



O(1)

$\frac{N}{2}$ + $\frac{N}{2}$) N

$\Theta(N \log N)$

work at each lvl times height

(c) Given that exponentialWork runs in $\Theta(3^N)$ time with respect to input N, what is the runtime of ronnie?

```
public void ronnie(int N) {
    if (N <= 1) {
        return;
    }
    ronnie(N - 2);
    ronnie(N - 2);
    ronnie(N - 2);
    exponentialWork(N); // Runs in Θ(3^N) time
}
```

$\Theta(3^N)$ as its the worst case out of all operations here

4 BST Asymptotics

Below we define the `find` method of a BST (Binary Search Tree) as in lecture, which returns the BST rooted at the node with key `sk` in our overall BST. In this setup, assume a `BST` has a `key` (the value of the tree root) and then pointers to two other child BSTs, `left` and `right`.

```
1  public static BST find(BST tree, Key sk) {
2      if (tree == null) {
3          return null;
4      }
5      if (sk.compareTo(tree.key) == 0) {
6          return tree;
7      } else if (sk.compareTo(tree.key) < 0) {
8          return find(tree.left, sk);
9      } else {
10         return find(tree.right, sk);
11     }
12 }
```

(a) Assume our BST is perfectly bushy. What is the runtime of a single `find` operation in terms of N, the number of nodes in the tree? Can we generalize the runtime of `find` to a tight bound?

log N

(b) Say we have an empty BST and want to insert the keys `[6, 2, 5, 9, 0, -3]` (in some order). In what order should we insert the keys into the BST such that the runtime of a single `find` operation after all keys are inserted is $O(N)$? Draw out the resulting BST.

we should insert them in decreasing or increasing order, so its growing linearly