CS 61B Spring 2021

More Asymptotics and ADTs

Exam Prep Discussion 7: March 1, 2021

1 Asymptotics is Fun!

(a) Using the function g defined below, what is the runtime of the following function calls? Write each answer in terms of N.

```
1  void g(int N, int x) {
2      if (N == 0) {
3          return;
4      }
5      for (int i = 1; i <= x; i++) {
6          g(N - 1, i);
7      }
8  }
```

g(N, 1): $\Theta($ $)$
g(N, 2): $\Theta($ $)$

Solution:
g(N, 1): $\Theta(N)$
g(N, 2): $\Theta(N^2)$

(b) Suppose we change line 6 to g(N - 1, x) and change the stopping condition in the for loop to i <= f(x) where f(x) returns a random number between 1 and x, inclusive. For the following function calls, find the tightest Ω and big O bounds.

```
1  void g(int N, int x) {
2      if (N == 0) {
3          return;
4      }
5      for (int i = 1; i <= f(x); i++) {
6          g(N - 1, x);
7      }
8  }
```

g(N, 2): $\Omega($ $)$, $O($ $)$
g(N, N): $\Omega($ $)$, $O($ $)$

Solution:
g(N, 2): $\Omega(N)$, $O(2^N)$
g(N, N): $\Omega(N)$, $O(N^N)$

2 Flip Flop

Suppose we have the `flip` function as defined below. Assume the method `unknown` returns a random `integer` between 1 and `N`, exclusive, and runs in constant time. For each definition of the `flop` method below, give the best and worst case runtime of `flip` in $\Theta(.)$ notation as a function of `N`.

```
1  public static void flip(int N) {
2      if (N <= 100) {
3          return;
4      }
5      int stop = unknown(N);
6      for (int i = 1; i < N; i++) {
7          if (i == stop) {
8              flop(i, N);
9              return;
10         }
11     }
12 }
```

(a)
```
public static void flop(int i, int N) {
    flip(N - i);
}
```

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

Solution:
Best Case: $\Theta(N)$, Worst Case: $\Theta(N)$

(b)
```
public static void flop(int i, int N) {
    int minimum = Math.min(i, N - i);
    flip(minimum);
    flip(minimum);
}
```

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

Solution:
Best Case: $\Theta(1)$, Worst Case: $\Theta(Nlog(N))$

(c)
```
public static void flop(int i, int N) {
    flip(i);
    flip(N - i);
}
```

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

Solution:
Best Case: $\Theta(N)$, Worst Case: $\Theta(N^2)$

3 Prime Factors

Determine the best and worst case runtime of `prime_factors` in $\Theta(.)$ notation as a function of `N`.

```
1  int prime_factors(int N) {
2      int factor = 2;
3      int count = 0;
4      while (factor * factor <= N) {
5          while (N % factor == 0) {
6              System.out.println(factor);
7              count += 1;
8              N = N / factor;
9          }
10         factor += 1;
11     }
12     return count;
13 }
```

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

Solution:
Best Case: $\Theta(log(N))$, Worst Case: $\Theta(\sqrt{N})$

4 ADT Selection

Suppose we have a TA Shreyas who teaches multiple discussion sections! A student may frequent more than one discussion section. For each situation below, choose the best ADT(s) out of the following — `Map`, `Set`, `List` — and explain how you can use the ADT(s) to solve the problem. Each subpart is independent of the previous. One answer may involve multiple ADTs. There may be mutliple efficient answers for each problem.

1. Storing all the `Students` in Shreyas's first section in alphabetical order.

2. Storing all the `Students` by their section, where `Students` within a section are sorted alphabetically.

3. Storing the `Students` in *all* of Shreyas's sections. There shouldn't be duplicates.

4. Determining all the `Students` that attend more than one of Shreyas's sections.

5. Quickly getting a `Student` by sid.

6. Quickly getting a `Student` by name *or* sid. Names aren't necessarily unique.

7. Cycling through the `Students` in one discussion section.

Solution:

1. Put the `Students` in a `List` in alphabetical order.
2. Use a `Map`, where each `Section` maps to an alphabetically ordered `List` of `Students` in that section.
3. Use a `Set`. Add all the `Students` to the `Set`.
4. Use two `Sets`. The first `Set` will store all the students seen so far, and the second `Set` will be our answer. Iterate through the `Students` in each `Section` and, if the first set contains the `Student`, add it to the second set. Otherwise, add the `Student` to the first set.

5. Use a `Map`, where each `sid` maps to one `Student`.
6. Use two `Maps`, where the first map works as described in the previous part and the second map maps `names` to a `Set` of `Students` of the given name.
7. Put the `Students` in a `List`, a `LinkedList` recommended. Repeat removing from the front and reinserting at the back.