CS 61B Spring 2022

Test 2 Review

Exam Prep Discussion 11: April 4, 2022

1 Round Down

Here is a video walkthrough of the solutions.

Given some power of two `powerOfTwo` and a positive number `num`, round `num` down to the nearest multiple of `powerOfTwo`. Assume `powerOfTwo` is greater than or equal to 1. You may use only bit operations and one subtraction/addition operation.

Examples:

```
roundDown(8, 53) -> 48
roundDown(16, 90) -> 80
roundDown(1, 90) -> 90
```

```
public int roundDown(int powerOfTwo, int num) {

   return ________________________________________________;
}
```

Solution:

```
public int roundDown(int powerOfTwo, int num) {
   return ~(powerOfTwo - 1) & num;
}
```

2 Heaps

a) (2.5 Points). i) (1 Point). Suppose we have the min-heap below (represented as an array) with **distinct** elements, where the values of A and B are unknown. Note that A and B aren't necessarily integers.

```
{1, A, 3, 5, 9, 11, 13, 10, B}
```

What can we say about the relationships between the following elements? Put >, <, or ? if the answer is not known.

A ◯ > ◯ < ◯ ? 1

A ◯ > ◯ < ◯ ? 3

B ◯ > ◯ < ◯ ? 10

A ◯ > ◯ < ◯ ? B

Solution:

Here is a video walkthrough of the solutions.

A ✓ > ◯ < ◯ ? 1

A ◯ > ◯ < ✓ ? 3

B ◯ > ◯ < ✓ ? 10

A ◯ > ✓ < ◯ ? B

ii) (1.5 Points). Note for both parts below, the values of `A` and `B` should **not** violate the min-heap properties. Put `-inf` or `inf` if there isn't a lower or upper bound, respectively. If the bound for `B` depends on the value of `A`, or vice versa, you may put the variable in the bound, e.g. `A < B`.

Considering **one `removeMin`** call, put **tight** bounds on `A` and `B` such that:

- We perform the **maximum** number of swaps.

  ------ < **A** < ------

  ------ < **B** < ------

- We perform the **minimum** number of swaps.

  ------ < **A** < ------

  ------ < **B** < ------

Solution:

Here is a video walkthrough of the solutions.

- We perform the **maximum** number of swaps.

  1 < A < 3

  10 < B < inf

- We perform the **minimum** number of swaps.

  3 < A < 5

  5 < B < 11

3 Hashing Asymptotics

Here is a video walkthrough of the solutions.

Suppose we set the `hashCode` and `equals` methods of the `ArrayList` class as follows.

```
/* Returns true iff the lists have the same elements in the same ordering */
@Override
public boolean equals(Object o) {
    if (o == null || o.getClass() != this.getClass() || o.size() != this.size()) {
        return false;
    }
    ArrayList<T> other = (ArrayList<T>) o;
    for (int i = 0; i < this.size(); i++) {
        if (other.get(i) != this.get(i)) {
            return false;
        }
    }
    return true;
}

/* Returns the sum of the hashCodes in the list. Assume the sum is a cached instance variable. */
@Override
public int hashCode() {
    return sum;
}
```

(a) Give the best and worst case runtime of `hashContents` in $\Theta(.)$ notation as a function of `N`, where `N` is initial size of the `list`. Assume the length of `set 's` underlying array is `N` and the `set` does **not** resize. Assume the `hashCode` of an `Integer` is itself. Admittedly, the `ArrayList` class does not have the method `removeLast`, but assume it does for this problem, and is implemented in amortized constant time. Finally, assume `f` accepts two `int`s, returns an unknown `int`, and runs in constant time.

```
static void hashContents(HashSet<ArrayList<Integer>> set, ArrayList<Integer> list) {
    if (list.size() <= 1) {
        return;
    }
    int last = list.removeLast();
    list.set(0, f(list.get(0), last));
    set.add(list);
    hashContents(set, list);
}
```

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

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

(b) Continuing from the previous part, how can we define `f` to **ensure** the worst case runtime? How can we define `f` to **ensure** the best case runtime? There may be multiple possible answers.

1. Worst case:

```
int f(int first, int last) {
    return ____________________________;
}
```

Solution:

```
int f(int first, int last) {
    return first + last;
}
```

2. Best case:

```
int f(int first, int last) {
    return ____________________________;
}
```

Solution:

```
int f(int first, int last) {
    return first + last + 1;
}
```

Alternate solution:

```
int f(int first, int last) {
    return first + last - 1;
}
```

4 Boolean Confusion

Here is a video walkthrough of the solutions.

Give the best and worst case runtime in $\Theta(.)$ notation as a function of `N`, where `N` is `arr.length`. Your answer should be simple with no unnecessary leading constants or summations.

```
1  void confusion(boolean[] arr) {
2      boolean first = arr[0];
3      int next;
4      for (next = 1; arr[next] == first; next++) {
5          if (next == arr.length - 1) {
6              return;
7          }
8      }
9      for (int i = 0; i < next; i++) {
10         arr[i] = !arr[i];
11     }
12     confusion(arr);
13 }
```

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

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

5 Gamma

Here is a video walkthrough of the solutions.

Give the best and worst case runtime in $\Theta(.)$ notation as a function of `N`. Your answer should be simple with no unnecessary leading constants or summations. Assume `f(N)` returns a random number between `1` and `N/2`, inclusive, and does so in constant time.

```
1  static void gamma(int N) {
2      if (N <= 10) {
3          return;
4      }
5      for (int i = f(N); i < N; i += f(N)) {
6          gamma(i);
7      }
8  }
```

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

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