Test 2 Review

Discussion 11 Slides

Announcements

- **Week 11 Survey** due Tuesday 4/5
- **Test 2** on Wednesday 4/6
- **Tournament Submission** due Sunday 4/10
- **Lab 12** due Monday 4/11

CS 61B Spring 2022 — Test 2 Review

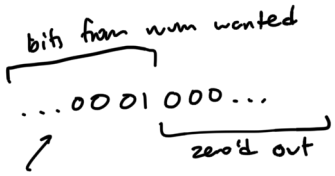
Exam Prep Discussion 11: April 4, 2022

1 Round Down

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
```



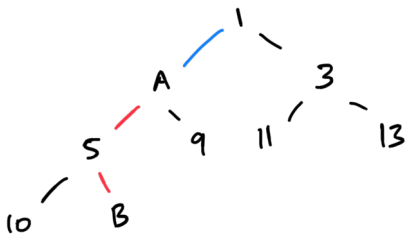

```
public int roundDown(int powerOfTwo, int num) {

    return ~(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 2 or 4 work

B ○ > ○ < ● ? 10 Many options

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.

Send B left, not right

from original → 1 < A < 3 Send B down left

10 < B < inf ← Can be as large as ∞

goes all the way left

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

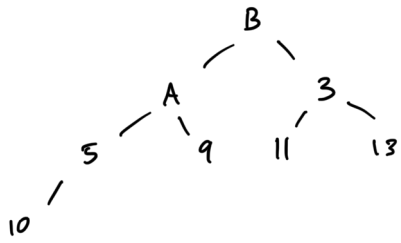
B swaps on the right → 3 < A < 5 Send B right by 1

5 < B < 11

still has to be valid

from original

only swap once



3 Hashing Asymptotics

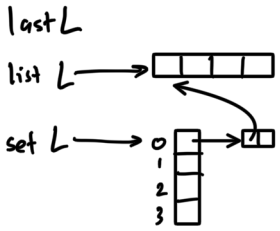
Suppose we set the `hashCode` and `equals` methods of the `ArrayList` class as follows.

```
/* 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;
}
```

sum of elements

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

go to different buckets (constant time insertion)

keep going to same bucket (.equals check takes N time)

(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:

```
1  int f(int first, int last) {
2      return first + last_________________;
3  }
```

keep the sum of the list to be the same
random doesn't ensure anything

2. Best case:

```
1  int f(int first, int last) {
2      return first + last  ± 1___________;
3  }
```

+ or - works here

decrement/increment the sum of the list
rotates around buckets (prevents any issues with periodicity)

4 Boolean Confusion

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 }
```

Stop condition is when a different boolean value is found

If whole array has the same value, best case enters here and doesn't do any recursion

All of the leading up to items are reset

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

Everything is the same value

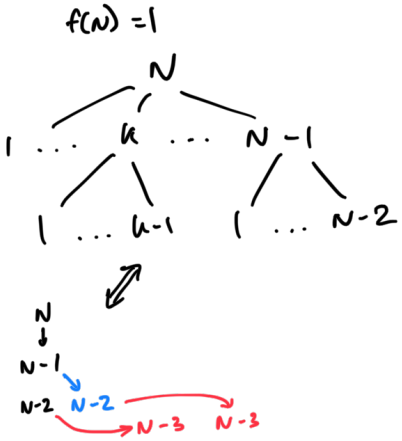
Array is flip-flopped TFTF...

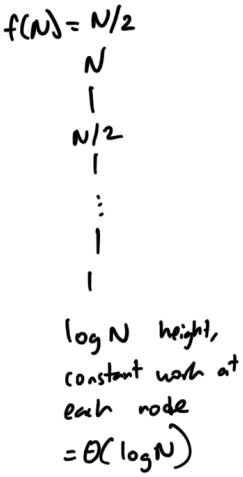
5 Gamma

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





log N height, constant work at each node
$= \Theta(\log N)$

Exponential growth!

$1 + 2 + 4 + \ldots + 2^N = 2^{N+1} - 1 = \Theta(2^N)$