CS 61BL Summer 2021

Final Exam Review

Monday August 9, 2021

1 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;
}
```

(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 the same as in Project 2. 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: $\Theta(N^2)$

Explanation:

last L3

list L → [?][2] 0 1

set →

0	→
1	/
2	→
3	/
4	→

Add :

10 1) compute hashcode ⟶ object.hashCode()

2) go to correct bucket → hashcode % M = bucket
(M = # of buckets)

3) Iterate through linked list
→ if .equal key, return
→ else insert at end of linked list

Two possibilities :

worst case: 1. go to bucket 0 : $\Theta(N)$

best case: 2. go to another bucket (that is empty) : $\Theta(1)$

WORST CASE:

$N-2 + N-3 + N-4 + \dots + 1$

$1+2+3+4+\dots+N$!

$\Theta(N^2)$

BEST CASE:

$1+1+\dots+1$

$\Theta(N)$

(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:

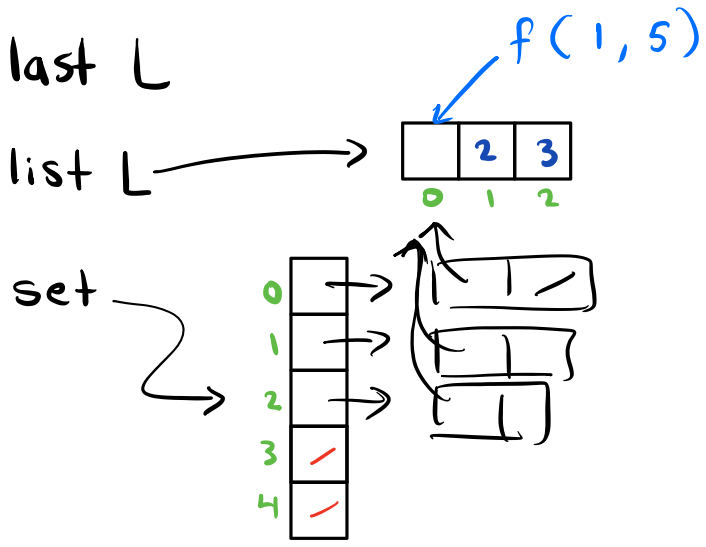
```
1  int f(int first, int last) {
2      return first + last;
3  }
```

2. Best case:

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

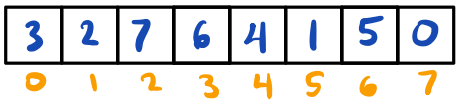
```
static void hashContents(HashSet<ArrayList<Integer>> set, ArrayList<Integer> list) {
    if (list.size() <= 1) {
        return;
    }
    int last = list.removeLast();
    list.set(0, f(list.get(0), last));
    set.add(list);
    hashContents(set, list);
}
```

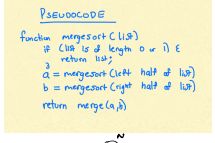
Explanation:

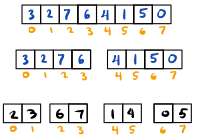


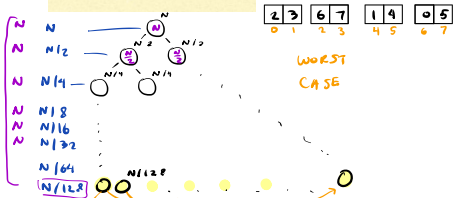
BEST CASE
→ add 1 to the sum as we recurse

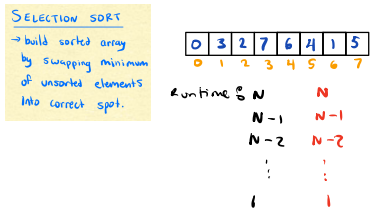
WORST CASE
→ go to the same bucket every time
→ our hashcode stays the same over time
→ sum of the list should never change

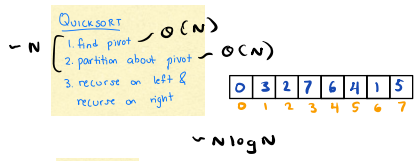


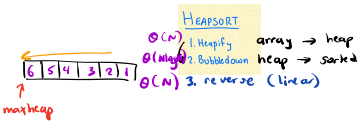












2 Sorted Runtimes

We want to sort an array of N **unique** numbers in ascending order. Determine the best case and worst case runtimes of the following sorts:

(a) Once the runs in merge sort are of *size* <= $N/100$, we perform insertion sort on them.

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

(b) We can only swap adjacent elements in selection sort.

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

(c) We use a linear time median finding algorithm to select the pivot in quicksort.

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

(d) We implement heapsort with a min-heap instead of a max-heap. You may modify heapsort but must maintain constant space complexity.

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

(e) We run an optimal sorting algorithm of our choosing knowing:

- There are at most N inversions

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

- There is exactly 1 inversion

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

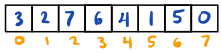
- There are exactly $(N^2 - N)/2$ inversions

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

[2 1 3 4 6 7 9 10 11]

idea: run insertion sort until we find the inversion

Recall an inversion is an "out of place" pair.
For instance, in the array:



→ (2,0) is an inversion since 2 > 0 but 2 precedes 0

→ (2,4) is not an inversion since 2 < 4 and 2 precedes 4

Runtime of insertion sort is $\Theta(N + k)$, where k = # of inversions

$\Theta(N + N)$

$\sim \Theta(N)$