Lecture 9: Order Statistics

Date: 09/19/2023

The i'th order statistics of a set of n elements is the i'th smallest element.
For example,

- the **minimum** of a set of elements is the 1'st order statistics (i=1)
- the **median** of a set of elements is the n+½'th order statistics
- the **maximum** of a set of elements is is the n'th order statistics

1'st Order Statistics

1_find_min.cpp

```
int find_min(vector<int> nums){
    int curr_min = INT_MAX;
    for(int i=0; i<nums.size(); i++){
        if(nums[i] < answer)
        answer = nums[i];
    }
    return answer;
}
```

⇒ Time Complexity: θ(n)
⇒ Space Complexity: θ(1)

2'nd Order Statistics

2_find_second_min.cpp

```
int find_second_min(vector<int> nums){
    int min_ele = INT_MAX;
```

```cpp
    int second_min = INT_MAX;

    for(int i=0; i<nums.size(); i++){
        // if current number is less than the min
        if(nums[i] < min_ele){
            // second min will be current min
            second_min = min_ele;
            // update the min
            min_ele = nums[i];
        }
        // if current number is greater than min
        // but less than second min
        else if(nums[i] < second_min){
            // update the second min
            second_min = nums[i];
        }
    }
    return second_min;
}
```

⇒ Time Complexity: θ(n)
⇒ Space Complexity: θ(1)

3’rd Order Statistics

3_find_third_min.cpp

```cpp
int find_third_min(vector<int> nums){
    int min_ele = INT_MAX;
    int second_min = INT_MAX;
    int third_min = INT_MAX;

    for(int i=0; i<nums.size(); i++){
        // if current number is less than the min
        if(nums[i] < min_ele){
            // third min will be current second min
            third_min = second_min;
            // second min will be current min
            second_min = min_ele;
```

```
            // update the min element
            min_ele = nums[i];
        }
        // if current number is greater than min
        // but less than second min
        else if(nums[i] < second_min){
            // third min will be current second min
            third_min = second_min;
            // update the second min
            second_min = nums[i];
        }
        else if(nums[i] < third_min){
            // update the third min
            third_min = nums[i];
        }
    }
    return third_min;
}
```

⇒ Time Complexity: θ(n)
⇒ Space Complexity: θ(1)

k'th Order Statistics

4_find_kth_min.cpp

```
void shift(vector<int>& nums, int from){
    for(int i=nums.size()-1; i>=from+1; i--){
        nums[i] = nums[i-1];
    }
}

int find_kth_min(vector<int> nums, int k){
    vector<int> mins(k, INT_MAX);

    for(int i=0; i<nums.size(); i++){
        // finding out if current num is less than kth min
        int index = -1;
        for(int j=0; j<k; j++){
            if(nums[i] < mins[j]){
                index = j;
```

```
                break;
            }
        }

        // if current number is greater than k'th min
        if(index == -1) continue;

        // update the other min elements
        shift(mins, index);

        // update correct min element
        mins[index] = nums[i];
    }

    return mins[k-1];
}
```

⇒ Time Complexity: θ(n*k)
⇒ Space Complexity: θ(k)

Selection Algorithm

5_selection_algorithm.cpp

```cpp
int find_kth_min(vector<int> nums, int k, int low, int high){
    if(low > high) return -1;
    int pos = partition(nums, low, high);

    int left_size = pos - low + 1;
    if(k < left_size)
        return find_kth_min(nums, k, low, pos-1);
    else if(k > left_size)
        return find_kth_min(nums, k-left_size, pos+1, high);

    return nums[pos];
}
```

Time Complexity Analysis

First Element as Pivot

⇒ If we always take first element as pivot, then the time complexity equation will be,
T(n) = T(x) + T(y) + θ(n)

So,

- Best Case: T(n) = θ(n * log(n)))
  Happens when every time the algorithm partitions the array in two equal parts.
  i.e. nums = [4,2,1,3,6,5,7]

- Worst Case: T(n) = θ(n * n)
  Happens when every time the algorithm partitions the array in one larger and one smaller part.
  i.e. nums = [7,6,5,4,3,2,1]

Last Element as Pivot

⇒ If we always take last element as pivot, then the time complexity equation will be,
T(n) = T(x) + T(y) + θ(n)

So,

- Best Case: T(n) = θ(n * log(n)))
  Happens when every time the algorithm partitions the array in two equal parts.
  i.e. nums = [4,2,1,3,6,5,7]

- Worst Case: T(n) = θ(n * n)
  Happens when every time the algorithm partitions the array in one larger and one smaller part.
  i.e. nums = [1,2,3,4,5,6,7]

Random Element as Pivot

⇒ If we take random element as pivot, then the time complexity equation will be,
T(n) = T(x) + T(y) + θ(n)

So,

- Best Case: T(n) = θ(n * log(n)))
  Happens when every time the algorithm partitions the array in two equal parts.
  i.e. nums = [4,2,1,3,6,5,7]

- Worst Case: T(n) = θ(n * n)
  Happens when every time the algorithm partitions the array in one larger and one smaller part.
  i.e. nums = [1,2,3,4,5,6,7]