Design Analysis and Algorithem, Lab Assignment 3

Bhuvana Kanakam - SE21UCSE035

September 7 , 2023

Problem Statement

Given a set of input data, you need to implement two different approaches to find the maximum and minimum values. The two approaches are:

1. **Divide and Conquer Approach:** Implement a method that uses the divide and conquer technique to find the maximum and minimum values within the input set.
2. **Without Divide and Conquer Approach:** Implement a method that finds the maximum and minimum values without using the divide and conquer technique.

Your task is to count the number of times the maximum and minimum values are updated during the execution of each approach. Additionally, you should vary the input data to test your implementations. Finally, determine which of the two approaches is better based solely on the number of updates to the maximum and minimum values and demonstrate this by plotting the complexity of these updates.

Approach and Analysis

For the Divide and Conquer Approach, we utilize a recursive strategy to divide the input set into smaller subproblems until we can find the maximum and minimum values.

Divide and Conquer Method

```
Algorithem
  MaxMin (i,j,max,min){
    if (i==j) then max,min = a[i] ; case of one element.
    else if (i = j-1) then ; case of two elements.
    {
      if (a[i] < a[j]) then
        max = a[i];
        min = a[j];
      else
        max = a[i];
        min = a[j];
    }
  else
  {
    mid = [(i+j)/2]
    MaxMin(i,mid,maxl,minl)
    MaxMin(mid+1,j,maxr,minr)
    if (maxl<maxr) then maxl = maxr;
    if (minl>minr) then minl = minr;
  }
}
```

Complexity : O(n)

But, it is not preferred to use this method cause of higher space complexity

For the Without Divide and Conquer Approach, we employ a straightforward iterative method to traverse the input set and update the maximum and minimum values.

Without Divide and Conquer Method

```
Straight Max Min (arn, min, max) {
  max = min = a[1]
  for i=2 to n do
    if (a[i]>max)
      then max = a[i]
    else if (a[i] <min)
      then min = a[i]
```

We keep track of the number of times the maximum and minimum values are updated for each approach to compare their efficiencies.

Answer

Below is the Python code for divide and conquer:

```
def max_min_divide_conquer_count_updates(arr, i, j):
    max_updates = 0
    min_updates = 0

    if i == j:
        return arr[i], arr[i], max_updates, min_updates
    elif i == j - 1:
        if arr[i] < arr[j]:
            max_updates += 1
            min_updates += 1
            return arr[i], arr[j], max_updates, min_updates
        else:
            max_updates += 1
            min_updates += 1
            return arr[j], arr[i], max_updates, min_updates
    else:
        mid = (i + j) // 2
        maxl, minl, maxl_updates, minl_updates = max_min_divide_conquer_count_updates(ar
        maxr, minr, maxr_updates, minr_updates = max_min_divide_conquer_count_updates(ar

        max_updates += maxl_updates + maxr_updates
        min_updates += minl_updates + minr_updates

        if maxl < maxr:
            maxl = maxr
            max_updates += 1
        if minl > minr:
            minl = minr
            min_updates += 1

        return maxl, minl, max_updates, min_updates
```

Below is the Python code for without divide and conquer:

```
def straight_max_min_count_updates(arr):
    if len(arr) == 0:
        return None, None, 0, 0

    max_val = min_val = arr[0]
    max_updates = 0
```

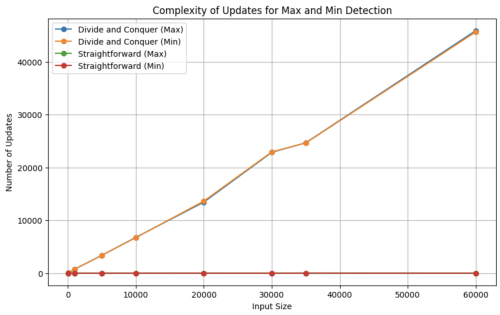

Figure 1: Enter Caption

```
min_updates = 0

for i in range(1, len(arr)):
    if arr[i] > max_val:
        max_val = arr[i]
        max_updates += 1
    elif arr[i] < min_val:
        min_val = arr[i]
        min_updates += 1

return max_val, min_val, max_updates, min_updates

input_sizes = [100, 1000, 5000, 10000, 20000, 30000, 35000, 60000]
update_counts_divide_conquer_max = []
update_counts_divide_conquer_min = []
update_counts_straight_max = []
update_counts_straight_min = []

for size in input_sizes:
input_data = [random.randint(1, 1000) for _ in range(size)]


_, _, max_updates_dc, min_updates_dc = max_min_divide_conquer_count_updates(input_da
update_counts_divide_conquer_max.append(max_updates_dc)
update_counts_divide_conquer_min.append(min_updates_dc)

_, _, max_updates_straight, min_updates_straight = straight_max_min_count_updates(in
update_counts_straight_max.append(max_updates_straight)
update_counts_straight_min.append(min_updates_straight)


plt.figure(figsize=(10, 6))
plt.plot(input_sizes, update_counts_divide_conquer_max, marker='o', label='Divide an
plt.plot(input_sizes, update_counts_divide_conquer_min, marker='o', label='Divide an
plt.plot(input_sizes, update_counts_straight_max, marker='o', label='Straightforward
plt.plot(input_sizes, update_counts_straight_min, marker='o', label='Straightforward
plt.xlabel('Input Size')
plt.ylabel('Number of Updates')
plt.title('Complexity of Updates for Max and Min Detection')
plt.legend()
plt.grid(True)
plt.show()
```