Name: Longchang Cui

ID: 16242D

Course: EN.605.621.81.SP21 Foundations of Algorithms

Programming Assignment #2

Generate Divide-and-Conquer recursion tree with Merge Sort

We can use merge sort algorithm to generate Divide-and-Conquer recursion tree. The following pseudo code does not include the recursive and non-recursive cost calculation script. The cost calculation code is in the file *Cui_PA2.py* from line 9 - line 82

Pseudo Code:

Merge-Sort(A)

1 **if** *A.length* > 1

2 mid = $\lfloor A.length / 2 \rfloor$

3 left = $A[1 \ldots \text{mid}]$

4 right = $A[\text{mid} \ldots A.length]$

5 *Merge-Sort(*left*)* *// Recursive call*

6 *Merge-Sort(*right*)* *// Recursive call*

7 i = 0, j = 0, k = 0

8 **while** i < left.*length* and j < right.*length* // Compare the element and swap

9 **if** left[i] < right[j]

10 A [k] = left[i]

11 i + 1

12 **else**

13 A [k] = right[j]

14 j + 1

15 k + 1

16 **while** i < left.*length* // Copy the remaining elements to array

17 A [k] = left[i]

```
18                  i + 1
19                  k + 1
20          while j < right.length          // Copy the remaining elements to array
21                  A [k] = right[j]
22                  j + 1
23                  k + 1
24 return A
```

Q1: What is the worst-case big-O asymptotic running time for your recursion tree generator algorithm?

The recursive call would take $O(\lg n)$ running time, and merging the elements will take $O(n)$

Thus, the total running time in worst case will take $O(\mathrm{n} \lg n)$

Q2: Does the worst-case big-O asymptotic running time vary depending on the recurrence function, and/or the f(n) forms?

The recurrence function of the merge sort is $T(n) = 2 * T(n/2) + \theta(n)$

Thus, the worst-case running time of the algorithm depends on $f(n)$ forms.

Python Code:

The merge sort python code is in the file *Cui_PA2.py* from line 9 - line 82

Q3: Accept polynomial f (n) forms?

Yes, the polynomial $f(n)$ form of the merge sort is $\theta(n)$

Q4: Generate the required four depths of the recursion tree?

Yes, the following input arrays were tested. The depth of each test case is four.

Array 1 = [13, 12, 10, 11, 15, 14, 16, 17]

Array 2 = [10, 11, 12, 13, 14, 15, 16, 17]

Array 3 = [17, 16, 15, 14, 13, 12, 11, 10]

Q5: Generate output representing all nodes at a depth in the recursion tree?

The following output is obtained from the IPython console with Python version 3.7.4.

The recursion tree is obtained before the sorting completes. To avoid the repetitiveness, only Array 1's output is presented. The complete output can be obtained by executing the file *Cui_PA2.py or open Cui_PA2_Output.txt file.*

Output for Array 1 = [13, 12, 10, 11, 15, 14, 16, 17]:

Current node's depth is 0. Array in current depth is [13 12 10 11 15 14 16 17]

Current node's depth is 1. Array in current depth is [13 12 10 11]

Current node's depth is 2. Array in current depth is [13 12]

Current node's depth is 3. Array in current depth is [13]

Current node's depth is 3. Array in current depth is [12]

Current node's depth is 2. Array in current depth is [10 11]

Current node's depth is 3. Array in current depth is [10]

Current node's depth is 3. Array in current depth is [11]

Current node's depth is 1. Array in current depth is [15 14 16 17]

Current node's depth is 2. Array in current depth is [15 14]

Current node's depth is 3. Array in current depth is [15]

Current node's depth is 3. Array in current depth is [14]

Current node's depth is 2. Array in current depth is [16 17]

Current node's depth is 3. Array in current depth is [16]

Current node's depth is 3. Array in current depth is [17]

Q6. Compute the recursive and non-recursive cost at each node?

The recursive and non-recursive cost is obtained from *Cui_PA2.py* and organized manually for readability. The recursive cost is obtained before the sorting, and the non-recursive cost is the value obtained after sorting completes. To avoid the repetitiveness, only Array 1's output is presented. The complete output can be obtained by executing the file *Cui_PA2.py or in Cui_PA2_Output.txt file.*

Cost output for Array 1 = [13, 12, 10, 11, 15, 14, 16, 17]:

Depth: 0, Array: [13 12 10 11 15 14 16 17], Recursive Cost: 0, Non-Recursive Cost after sort: 8

Depth: 1, Array: [13 12 10 11], Recursive Cost: 0, Non-Recursive Cost after sort: 4

Depth: 2, Array: [13 12], Recursive Cost: 0, Non-Recursive Cost after sort: 2

Depth: 3, Array: [13], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 3, Array: [12], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 2, Array: [10 11], Recursive Cost: 0, Non-Recursive Cost after sort: 2

Depth: 3, Array: [10], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 3, Array: [11], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 1, Array: [15 14 16 17], Recursive Cost: 0, Non-Recursive Cost after sort: 4

Depth: 2, Array: [15 14], Recursive Cost: 0, Non-Recursive Cost after sort: 2

Depth: 3, Array: [15], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 3, Array: [14], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 2, Array: [16 17], Recursive Cost: 0, Non-Recursive Cost after sort: 2

Depth: 3, Array: [16], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Depth: 3, Array: [17], Recursive Cost: 0, Non-Recursive Cost after sort: 0

Q7. Compute the total non-recursive cost at each depth of the recursion tree in LCD form?

The total non-recursive cost is obtained from *Cui_PA2.py* and organized manually for readability. To avoid the repetitiveness, only Array 1's output is presented. The complete output can be obtained by executing the file *Cui_PA2.py or in Cui_PA2_Output.txt file.*

Cost output for Array 1 = [13, 12, 10, 11, 15, 14, 16, 17]:

Depth: 0, Array: [13 12 10 11 15 14 16 17], Total-Non-Recursive Cost after sort: 8

Depth: 1, Array: [13 12 10 11], [15 14 16 17], Total-Non-Recursive Cost after sort: 8

Depth: 2, Array: [13 12], [10 11], [15 14], [16 17] Total-Non-Recursive Cost after sort: 8

Depth: 3, Array: [13], [12], [10], [11], [15], [14], [16], [17], Total-Recursive Cost after sort: 0

Retrospect:

The recursive and the non-recursive cost of the merge sort algorithm is the same as long as we have the same size of the input array. Regardless of what values the input array has, we would still run the algorithm even if the input array is already sorted.

Generate Chip-and-Be-Conquered recursion tree with Heap Sort

We can use heap sort algorithm to generate Chip-and-Be-Conquered recursion tree. The following pseudo code does not include the heapify and the cost calculation script. The complete cost calculation, heapify, and the building max-heap code is in the file *Cui_PA2.py* from line 111 - line 192

Pseudo Code:

Heap-Sort(A)

1 Max-Heap(*A*) // Build max-heap

2 **for** i = *A.length* **down to** 2

3 Swap(*A[j], A[1])* // Swap

4 *A.length = A.length - 1*

5 heapify(A, i) // Heapify to sort the element

Q1: What is the worst-case big-O asymptotic running time for your recursion tree generator algorithm?

If we only consider the sorting, the heapfy call would take $O(\lg n)$ running time, and looping through elements in an array will take $O(n-1) = O(n)$

Thus, the total running time in worst case will take $O(\mathrm{n} \lg n)$

Q2: Does the worst-case big-O asymptotic running time vary depending on the recurrence function, and/or the f(n) forms?

The recurrence function of the heap sort is $T(n) = T(n-1) + O(\lg n)$

Thus, the worst-case running time of the algorithm depends on recurrence function.

Python Code:

The heap sort python code is in the file *Cui_PA2.py* from line 111 - line 192

Q3: Accept polynomial f (n) forms?

Yes, the polynomial $f(n)$ form of the heap sort is $O(\lg n)$

Q4: Generate the required four depths of the recursion tree?

Yes, the following input arrays were tested. We are building a max-heap before the sorting, and the depth of max-heap on each test case is four.

Array 1 = [13, 12, 10, 11, 15, 14, 16, 17, 19, 18, 20, 21, 23, 22, 24]

Array 2 = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

Array 3 = [24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10]

Q5: Generate output representing all nodes at a depth in the recursion tree?

The following output is obtained from the IPython console with Python version 3.7.4. To avoid the repetitiveness, only Array 3's output is presented and not all output logs are presented. The complete output can be obtained by executing the file *Cui_PA2.py* or from the *Cui_PA2_Output.txt* file.

Because each recursive call, we pass the entire max-heap to the heapify method, the recursion tree will contain different values on each iteration.

Initial Max-heap of input Array 3 = [24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10]:

Iteration 1:

Recursive heapify call 1:

Max-Heap Value: [10, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 24]

Recursive heapify call 2:

Max-Heap Value: [23, 10, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 24]

Recursive heapify call 3:

Max-Heap Value: [23, 21, 22, 10, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 24]

Recursive heapify call 4:

Max-Heap Value: [23, 21, 22, 17, 20, 19, 18, 10, 16, 15, 14, 13, 12, 11, 24]

.....

.....

.....

Iteration 14:

Recursive heapify call 1:

Max-Heap Value: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

Q6. Compute the recursive and non-recursive cost at each node?

The recursive and non-recursive cost is obtained from *Cui_PA2.py* and organized manually for readability.

Cost output for Array 3 = *[24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10]:*

The recursive cost by each iteration is :

Iteration 1: recursive-cost 4

Iteration 2: recursive-cost 4

Iteration 3: recursive-cost 4

Iteration 4: recursive-cost 4

Iteration 5: recursive-cost 3

Iteration 6: recursive-cost 3

Iteration 7: recursive-cost 3

Iteration 8: recursive-cost 3

Iteration 9: recursive-cost 2

Iteration 10: recursive-cost 2

Iteration 11: recursive-cost 2

Iteration 12: recursive-cost 2

Iteration 13: recursive-cost 2

Iteration 14: recursive-cost 1

The non-recursive cost of each iteration is 1.

Q7. Compute the total non-recursive cost at each depth of the recursion tree in LCD form?

The total non-recursive cost is obtained from *Cui_PA2.py* and organized manually for readability. To avoid the repetitiveness, only Array 3's output is presented. The complete output can be obtained by executing the file *Cui_PA2.py or in Cui_PA2_Output.txt file.*

Cost output for Array 3 = [24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10]:

The total heap recursive cost is 39.

The total non-recursive cost is 14.

Retrospect:

Regardless of what values the input array has, we would still run the algorithm even if the input array is already sorted. Thus, the best-case, average-case, and the worst-case of the recursive algorithm would still be $O(\mathrm{n} \lg n)$

References:

Heap Sort Algorithm. (n.d.). Programiz. Retrieved March 14, 2021, from https://www.programiz.com/dsa/heap-sort

Recurrence Relations. (n.d.). Indian Institute of Information Technology. Retrieved March 14, 2021, from http://iiitdm.ac.in/old/Faculty_Teaching/Sadagopan/pdf/DAA/new/recurrence-relations-V3.pdf

Recurrences. (n.d.). University of Verginia. Retrieved March 14, 2021, from http://www.cs.virginia.edu/~horton/cs4102/page4/files/04-recurrences-divconq.ppt.pdf

Heap Sort. (n.d.). Washington State University. Retrieved March 14, 2021, from https://eecs.wsu.edu/~cook/aa/lectures/l3.pdf