CS566Lab3_2024

September 17, 2024

0.1 Lab 3. Introduction to algorithms

This is third Lab for CS 566. This problem was given in lecture.

0.2 Task 1. Solve the problem "Sort an Array" from https://leetcode.com/problems/sort-an-array/description/ using Python3. You are only allowed to use Merge Sort algorithm. You need to implement it yourself.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[10]:
```python
from typing import List

class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        # implement working algorithm
        if len(nums)>1:
            # split to left right
            left_arr = nums[:len(nums)//2]
            right_arr = nums[len(nums)//2:]

            # recursion
            self.sortArray(left_arr)
            self.sortArray(right_arr)

            i,j,k = 0,0,0 # left, right, merged idx
            # merge the sorted arrays
            while i<len(left_arr) and j<len(right_arr):
                if left_arr[i] < right_arr[j]:
                    nums[k] = left_arr[i]
                    i+=1
                else:
                    nums[k] = right_arr[j]
                    j+=1
                k+=1
            # add leftovers
            while i<len(left_arr):
                nums[k] = left_arr[i]
```

```
                i+=1
                k+=1
            while j<len(right_arr):
                nums[k] = right_arr[j]
                j+=1
                k+=1
        return nums
```

0.2.1 Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

[11]:
```
#test_case_1
expected, nums = [1,2,3,5], [5,2,3,1]
actual = Solution().sortArray(nums)
assert expected==actual, "Mistake in test case 1"

#test_case_2
expected, nums = [0,0,1,1,2,5], [5,1,1,2,0,0]
actual = Solution().sortArray(nums)
assert expected==actual, "Mistake in test case 2"
print('OK')
```

```
OK
```

0.2.2 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(nlogn)

Invariant: Merge sort relies on the invariant that the subarrays being merged are always sorted. Initially, it divides the array into smaller subarrays until each contains a single element, which is trivially sorted. As it merges these subarrays, it ensures that the merged result is sorted by comparing the smallest elements. This systematic maintenance of sorted order guarantees that the final array is completely sorted.

0.2.3 Task 2. Theoretical problem (5 points)

Use Master Theorem to prove that Merge Sort has algorithmic complexity O(nlogn). You may use latex in colab, or write the proof on paper and embed screenshot into colab. Make sure to use mathematical symbols as this is **theoretical problem**

Your answer goes here

[12]:
```
from IPython.display import display, Image
display(Image(filename="proof.png"))
```

$2T(n/2) + O(n), a = 2, b = 2$

Case 2:

$f(n) = \Theta(n^{log_a b}) = \theta(n^{\log_2 2}) = \Theta(n)$

Therefore:

$T(n) = \Theta(n^{log_a b} log(n)) = \Theta(n^{log_2 2} log(n)) =$

$\Theta(nlogn)$

0.2.4 Task 3. Theoretical problem (5 points)

Use Stirling's Approximation to prove that for **ANY** comparison-based algorithm the lower bound of Asymptotic Complexity is $n * log(n)$, meaning there **does not exist** a comparison-based sorting algorithm that can sort faster than $n * log(n)$.

Formally speaking you must prove that $time(n) = \Omega(n*log(n))$ $

Your answer goes here According to decision tree computational, there are $n!$ outcomes. See below diagram for example of 3 element array decision tree.

Since there are n! outcomes and the worst case number of comparisons in the decision tree is h (height of tree), then there are at most 2^h nodes in the tree, giving us:

$2^d \geq n!$

Simplify:

$h \geq \log_2 n!$

Apply Stirling Approximation:

$n! \sim 2\pi n \ (e/n \ )^n$

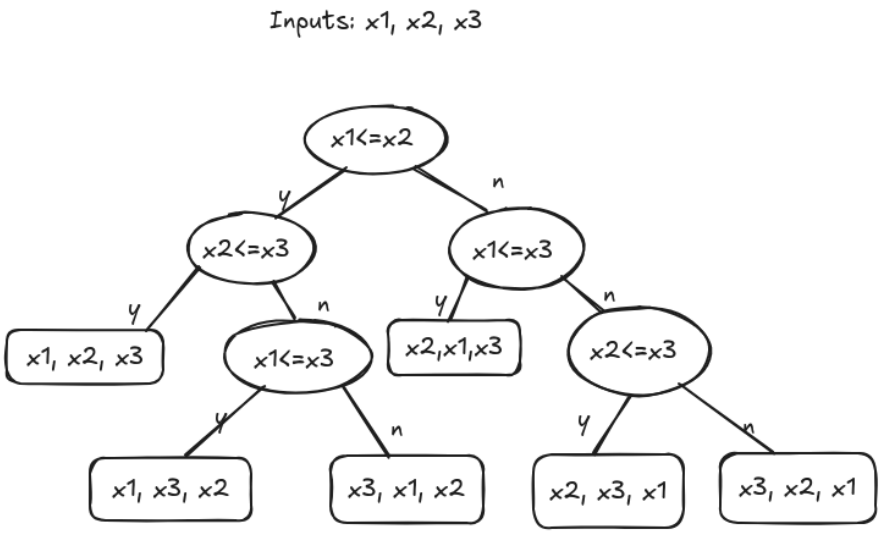
$\log_2(n!) \approx n \log_2(n) - n \log_2(e) + \frac{1}{2} \log_2(2\pi n)$

$\log_2(n!) = \Theta(n \log_2(n))$

Therefore:

$h \geq \Theta(n \log_2(n))$

[13]:
```
from IPython.display import display, Image
display(Image(filename="decisiontree.png"))
```



0.2.5 Task 4. Implement Stalin Sort (4 points)

This algorithm is not a serious algorithm, but it has good educational purposes. The algorithm starts from the first element and compares it with the second. If the second element is smaller than first, the algorithm deletes the second element and moves forward.

[14]:
```
ls = [1, 4, 2, 8, 11, 13, 15, 100, 39, 45, 50]
```

[15]:
```
def stalin_sort(ls):
    # your code goes here
    result = [ls[0]]
    for i in range(1,len(ls)):
      if ls[i]>=result[-1]:
        result.append(ls[i])
    return result
```

[16]:
```
expected = [1, 4, 8, 11, 13, 15, 100]
actual = stalin_sort(ls)
assert expected==actual, "Mistake in test case 1"
print('OK')
```

```
OK
```

0.2.6 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n)

Invariant: First element in original array will always be added to final array. Only elements in original array larger than first element in final array will be appended to final array.