Recusrion (Advance)

November 23, 2022

1 Array Algorithm (Recursive)

1.1 1. Binary Search with recursion

Binary search works when we have a sorted list, and we are searching for a value. We first look at middle and see if it is the element we are looking for. If it is the problem is solved If it is larger than the target element, then we re-run the search algorithm in the left part If it is smaller than the target element, then we re-run the search algorithm in the right part So the worst case Time complexity is O(n2)

[1]:
```
def rec_binary_search(nums,x,start,end):
    if start>end:
        return -1
    mid = start + (end-start)//2
    if nums[mid]==x:
        return mid
    elif nums[mid]>x:
        return rec_binary_search(nums,x,start,mid-1)
    return rec_binary_search(nums,x,mid+1,end)
```

[2]:
```
def search_fast(nums,x):
    return rec_binary_search(nums,x,0,len(nums))
```

[3]:
```
print(search_fast([8,13,19,23,35,45,65,78,89,99],89))
print(search_fast([8,13,19,23,35,45,65,78,89,99],12))
```

```
8
-1
```

1.2 2. Merge Sort

Merge Sort is another one of divide-and-conquer algorithms. Here we sort the array by dividing in into two halves, then sort both the halves, and then merge the sorted half togather.

[4]:
```
def merge(left,right,nums):
    left.append(float('inf'))
    right.append(float('inf'))
    i = 0
    j = 0
    k = 0
    while i<len(left)-1 or j<len(right)-1:
        if left[i] < right[j]:
            nums[k] = left[i]
            i+=1
        else:
            nums[k] = right[j]
            j+=1
        k+=1
    return
```

[5]:
```
def merge_sort(nums):
    if len(nums)==1 or len(nums)==0:
        return nums
    mid = len(nums)//2
    left = nums[:mid]
    right = nums[mid:]
    merge(merge_sort(left),merge_sort(right),nums)
    return nums
```

[6]:
```
nums_list = [98,18,23,67,23,54,17,11,9,100,39]
merge_sort(nums_list)
print(nums_list)
```

```
[9, 11, 17, 18, 23, 23, 39, 54, 67, 98, 100]
```

1.3 3. Quick Sort

Quick Sort is also a divide-and-conquer algorithm We keep an element Take it to its right position make all the tings left to it, smaller than it and right to it, larger than it then re-run the algorithm on that smaller parts

[5]:
```
def partition(nums,start,end):
    p = nums[start]
    k = 0
    for i in range(start,end+1):
        if nums[i]<p:
            k+=1
    index = start+k
    nums[start],nums[index] = nums[index],nums[start]
    i = start
    j = end
    while i < index < j:
        if nums[i]<p:
            i+=1
        elif nums[j]>p:
            j-=1
        else:
            nums[i],nums[j] = nums[j],nums[i]
            i+=1
            j-=1
    return index
```

[6]:
```
def quick_sort(nums,start,end):
    if start>=end:
        return
    i = partition(nums,start,end)
    quick_sort(nums,start,i-1)
    quick_sort(nums,i+1,end)
```

[7]:
```
nums_list = [98,18,23,67,23,54,17,11,9,100,39]
quick_sort(nums_list,0,len(nums_list)-1)
print(nums_list)
```

```
[9, 11, 17, 18, 23, 23, 39, 54, 67, 98, 100]
```