Justin Ciocoi

Nov. 2, 2023

CSCI 377 Video Notes

Chapter 7: Quick Sort

Quick Sort Algorithm

- This is an algorithm used to sort an array of numbers'
- An array is considered *sorted* when every element of the array is in its *correct position*, meaning all the numbers to each element's left is smaller than the element, and all the numbers to each element's right is larger
- When every single element is in its *correct position* we can consider the array *sorted*

Description of Quick Sort

- **Divide:** Divide the array $A[p, ..., r]$ into two sub-arrays, $P_1 = A[p, ..., q-1]$ and $P_2 = A[q+1, ..., r]$ such that each element of $P_1 \leq A[q]$, and $P_2 \geq A[q]$
- **Conquer**: Sort P_1 and P_2 by recursively calling quick sort
- **Combine:** No more work is needed because the sub-arrays are already sorted

Example

0	1	2	3	4	5	6
8	2	4	6	3	7	*5*
p						r

$$\downarrow$$

0	1	2	3	4	5	6
2	4	3	*5*	8	6	7

- In this example, element 5 is $A[q]$, which is known as the *pivot*

Partition Algorithm

- Given an array, choose any of its elements (generally the last element) and call it the *pivot*
- Move the elements less than the pivot to its left and the elements greater than the pivot to its right
- This operation will place the pivot at its *correct position*

```
Partition(A, p, r)
    x=A[r]
    i=p-1
    for j=p to r-1
        if A[j]<=x
            i=i+1
            swap(A[i], A[j])
    swap(A[i+1], A[r])
    return i+1
```

- The parameters passed to the partition function are A, which is the array of elements, p, which is the index of the first element in the sort operation, and r, which is the last element in the sort operation

Example

0	1	2	3	4	5	6
8	2	4	6	3	7	5
j						r=pivot

- **1st for loop:**
  - $x = A[r] = 5$
  - $i = -1$
  - $j = 0$
  - is $A[0] \leq x$
    - No
  - $j++$
- **2nd for loop:**
  - $i = -1$

    - $j = 1$
    - is $A[1] \leq x$
        - Yes
        - $i++$
        - $swap(A[0], A[1])$
    - $j++$

$$\downarrow$$

0	1	2	3	4	5	6
2	8	4	6	3	7	*5*
i		j				r=pivot

- 3rd for loop:
    - $i = 0$
    - $j = 2$
    - is $A[2] \leq x$
        - Yes
        - $i++$
        - $swap(A[1], A[2])$
    - j++

$$\downarrow$$

0	1	2	3	4	5	6
2	4	8	6	3	7	*5*
	i		j			r=pivot

- 4th for loop:
    - $i = 1$
    - $j = 3$

- is $A[3] \leq x$
    - No
- $j++$

- **5th for loop:**
  - $i = 1$
  - $j = 4$
  - is $A[4] \leq x$
    - Yes
    - $i++$
    - $swap(A[2], A[4])$
  - $j++$

$$\downarrow$$

0	1	2	3	4	5	6
2	4	3	6	8	7	5
		i			j	r=pivot

- **6th for loop (last one):**
  - $i = 2$
  - $j = 5$
  - is $A[5] \leq x$
    - No
  - $j++$
- **Exit for loop:**
  - $swap(A[i+1], A[r]) == swap(A[3], A[6])$
  - Return $i+1$, or the index of the correct position for pivot x

$\downarrow$

0	1	2	3	4	5	6
2	4	3	5	8	7	*6*
		i				j=r=pivot

- Now, we can finish sorting this array recursively, when we consider the two sub-arrays: $P_1 = [2, 4, 3]$, and $P_2 = [8, 7, 6]$
- These three-element arrays will be fully sorted by the partition operation, and since the pivot between the two sub-arrays is in the correct position, the entire array would now be sorted
- For example, after doing `partition(A, 0, 6)`, the user can recursively call `partition(A, 0, 2)` and `partition(A, 4, 6)` to finish sorting this array