APS 105 Lecture 35 Notes

Last time: selection sort, bubble sort and introduced quicksort

Today: Recap quicksort and develop its source code

Recap:

E.g. 2 1 $\boxed{3}$ 5 4

sorted as it is in its correct position, also called as "pivot" where all #s before are <3 and all #s after are >3.

Quicksort works by ordering a pivot in its location, then work on left subarray then on right subarray.

Example:

pivot = 10

0	1	2	3	4	5	6	7	8
left								right
10	14	8	13	20	3	6	9	4

A[right]<10 From right, look at number that belongs to the left of pivot

left++ From left, look at number that belongs to the right of pivot

swap 14 & 4

0	1	2	3	4	5	6	7	8
	left							right
10	4 ~~14~~	8	13	20	3	6	9	~~4~~ 14

Increment left till you find A[left] > pivot

Decrement right till you find A[right] < pivot

swap 13 & 9

0	1	2	3	4	5	6	7	8
			left				right	
10	4	8	~~13~~ 9	20	3	6	~~9~~ 13	14

0	1	2	3	4	5	6	7	8
			left				right	
10	4	8	9	20	3	6	13	14

Increment left till you find A[left] > pivot

Decrement right till you find A[right] < pivot

Swap 20 & 6

0	1	2	3	4	5	6	7	8
				left		right		
10	4	8	9	~~20~~ 6	3	~~6~~ 20	13	14

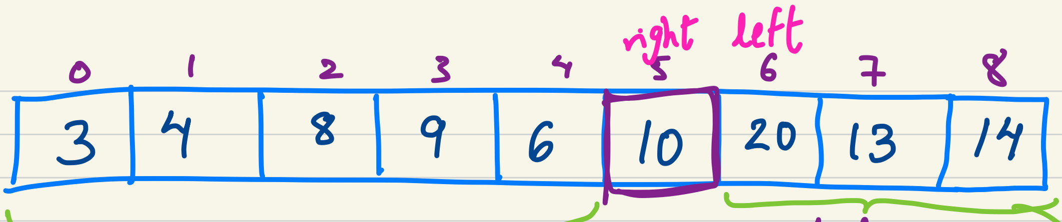
Increment left till you find A[left] > pivot

Decrement right till you find A[right] < pivot

Now (left > right)

0	1	2	3	4	5	6	7	8
					right	left		
~~10~~ 3	4	8	9	6	~~3~~ 10	20	13	14

Exchange pivot with A[right]



10 is now sorted!

all elements on left is < 10
all elements on right > 10

Continue doing Quicksort on left sub-array

Continue doing QuickSort on right sub-array

Step 1: implement a function that works on an array from index low to index high. Finds a pivot and makes all #s before it < pivot and all # after it > pivot. The function returns the final index of the pivot.

```
int partition(int list[], int low, int high){
    bool done = false;
    int pivotInd = low, left = low+1, right = high;
    while( !done ){
        while(left<=right && list[pivotInd]>list[left])
            left++;
        While(left<=right && list[pivotInd] < list[right])
            right--;
        if(left < right) swap(&list[left], &list[right]);
        else { swap(&list[right], &list[pivotInd]);
               done = true; }
    return right;
}
```

I want left & right to cross one another to stop!

Step 2: Call partition but on the subarray on the left of pivot and right of pivot

```
void quickSort(int list[], int length){
    quickSortHelper(list, 0, length-1);
}
void quickSortHelper(int list, int low, int high){
    if(low < high){
        int pivotInd = partition(list, low, high);
```

Sorts the left subarray

```
        quickSortHelper(list, low, pivotInd -1);
```

Sorts the right subarray

```
        quickSortHelper(list, pivotInd+1, high);
    }
}
```

Base Case is to do nothing!

Searching Algorithms

Search for an element in an array

```
int sequentialSearch ( int list[], int length, int data){
        int index = -1;
        for ( int i=0; i< length && index==-1; i++)
            if ( list[i] == data)
                index = i;
    return i;
}
```

At most we do length comparisons

Best case : 1 comp.

Average: $n/2$ comp.

Is there a better way?

If my array was sorted, e.g.

0	1	2	3	4
1	3	5	10	13

and I want to look for 10?

1) look at $arr[n/2] = arr[5/2] = arr[2] = 5$

if $10 > 5 \rightarrow$ look right subarray

$10 < 5 \rightarrow$ look left subarray

2) Repeat (1)

$$arr\left[\frac{n/2 + n}{2}\right] = arr[3] = 10$$

if $10 == 10 \rightarrow$ found

We eliminate half of the array everytime.

We do $\log_2(\text{length})$ comparisons.

We call the method "binary search"

Iterative Solution

```
int binarySearch(int list[], int length, int data){
        int low = 0, high = length-1;
        while( low <= high ){
                int middle = (low + high)/2;
                if ( list[middle] == data)
                        return middle;
                else if( list[middle] > data)
                        high = middle - 1;
                else
                        low = middle + 1;
        }
}
```

0 1 3 5 8 13 look for 1

① low = 0 middle = 2 high = 5

② low = 0 middle = 0 high = 1

③ low = 1 middle = 1 high = 1 found, so we need to enter loop when low == high

Recursive

```
int binarySearchHelper(int list[], int length,
                       int data, int low, int high){
    if (low > high)
        return -1;

    int middle = (low + high)/2;
    if (list[middle] == data)
        return middle;
    if (list[middle] > data)      go left
        return binarySearchHelper(list, length, data,
                                  low, middle-1);
    else                          go right
        return binarySearchHelper(list, length, data,
                                  middle+1, high);
}

int binarySearch(int list[], int length, int data){
    return binarySearchHelper(list, length, data, 0,
                              length-1);
```