# APS105 Lecture 34 Notes

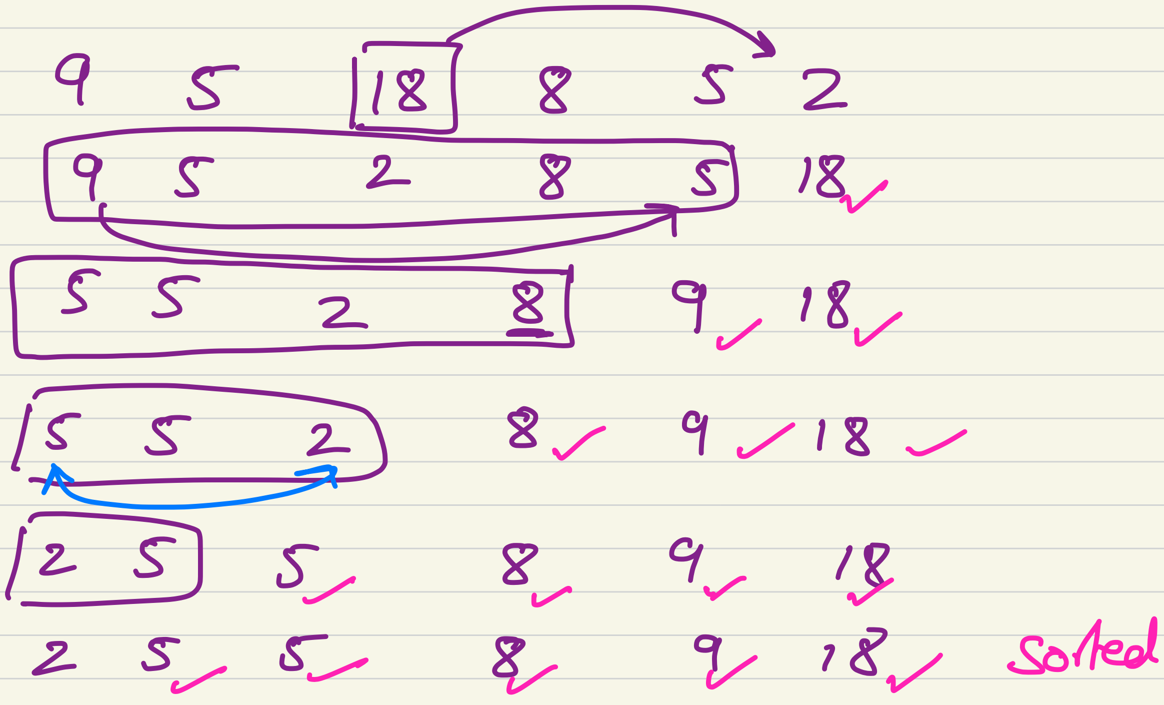
Last time: Concluded discussions on delete functions on linked list and discussed insertion sort.

Today: Discuss selection sort, bubble sort and introduce quick sort

Recall: We discuss our sorting algorithms on int arrays and we sort them in ascending order. Code we discuss can be amended to sort in descending order or on a linked list, for example.

## Selection Sort

- Search entire array to find largest & move it to the end (swap with end).
- Then search for largest element excluding last element, since it is in the correct place



How many times did we look for the largest #?

size of array -1

How much work in each time we search?

1st time : 6
2nd time : 5
⋮
last time : 2

```
void   selectionSort( int list[], int n){
       int top, large Loc, i;
       for ( int top = n-1; top > 0; top -- ){
           Large Loc = 0; //assume 1st element is largest
           for(int i = 1; i<= top; i++)
               if( list[i] > list[large Loc]){
                   Large Loc = i;
               }
       }
       //swap largest element found with top.
       to be placed in right place
       int temp = list[top];
       list[top] = list[Large Loc];
       list[Large Loc] = temp;
    }
    return;
}
```

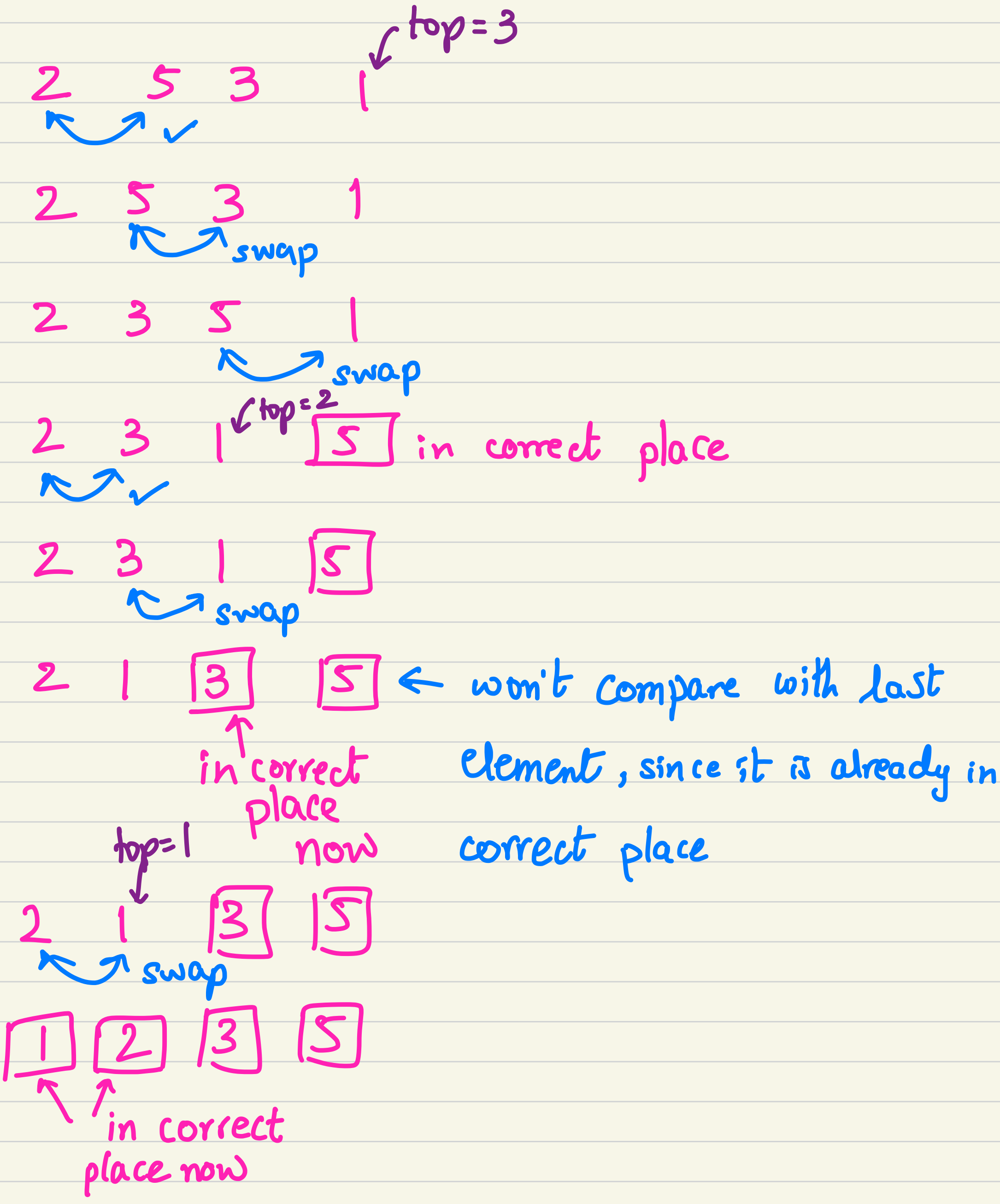
index of largest element in 1 iteration

look for largest element to put in index top

# Bubble sort

1) Go from left to right
2) Compare two elements next to each other & swap if you found them out of order
3) You will find largest element bubbled its way to the right
4) Repeat (1)

E.g



← won't compare with last element, since it is already in correct place

Design: ① 1 loop to go from left to right till the correctly placed element
② 1 loop to repeat ① until array is sorted

size of array

index before the correctly placed element

do comparisons till correctly placed element

```
void bubbleSort( int list[], int n){
        bool sorted = false;

        for (int top = n-1; top > 0&&!sorted, top--)
            sorted = true;
            for (int i= 0;  i < top ;  i++){
                    if ( list[i] > list[i+1]) {
                        int temp = list[i];
                        list[i]= list[i+1];
                        list [i+1]= temp;
                        sorted = false;
                    }
            }
```

will not do any swaps if list is sorted!

What if array is sorted ?

E.g. 1 2 3 4

will not do swaps in inner loop if array is sorted, hence not need to continue with outer loop.

## Quicksort –

E.g. 1  $\boxed{1}$ 3 2 6 5

is sorted because it's in its correct position

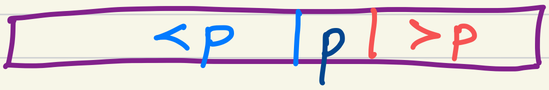
E.g. 2  3 8 1 0 $\boxed{9}$

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

not sorted

sorted (it's in its correct position) because all elements on its left are smaller than 3, all elements on its right are bigger than 3

Quicksort chooses one element and name it "pivot" and make all elements on its left smaller than the pivot and all element on its right larger.



pivot is now sorted!

→ Repeat algorithm on the left subarray and right subarray

Example:

pivot = 10

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| left | | | | | | | | right |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 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

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | left | | | | | | | right |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 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

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

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | left | | | | right | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 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

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

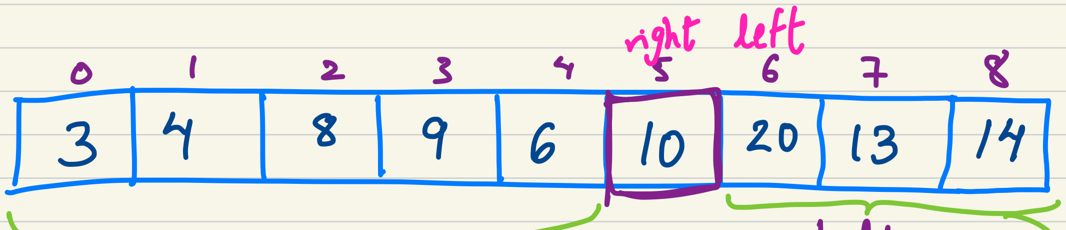
Increment left till you find A[left] > pivot

Decrement right till you find A[right] < pivot

Now (left > right)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | right | left | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ~~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

Can do this recursively, which we discuss next lecture!