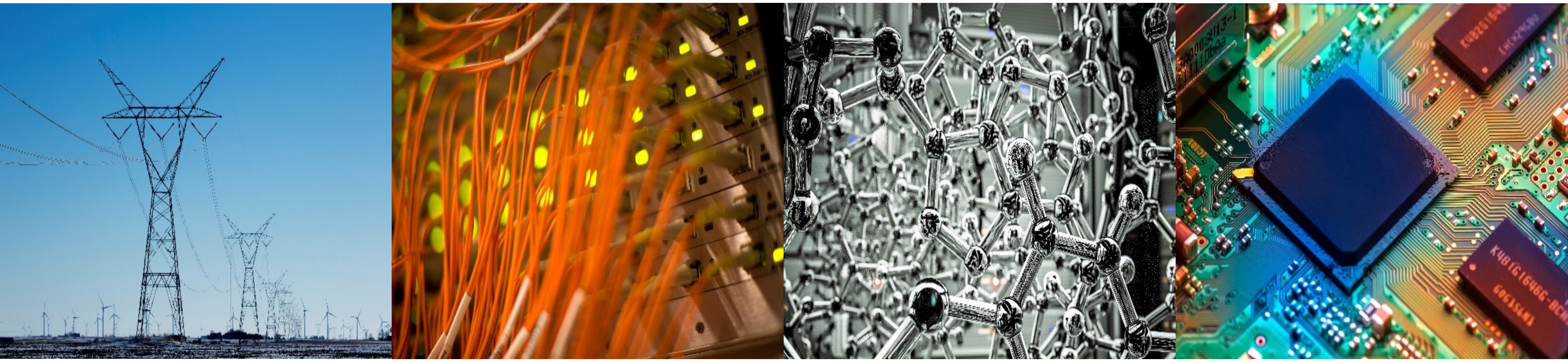
ECE 220 Computer Systems & Programming

Lecture 12 – Sorting Algorithms & Recursion

March 3, 2026



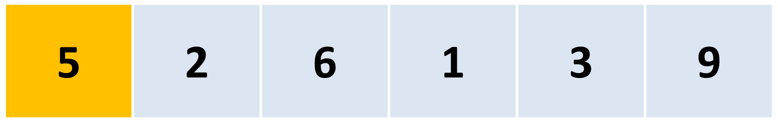
- Quiz2 should be completed at CBTF this week

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

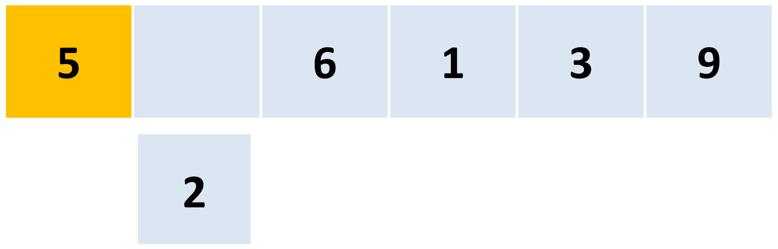
Insertion Sort:

1. remove item from array, insert it at the proper location in the sorted part by shifting other items;
2. repeat this process until the end of array is reach.

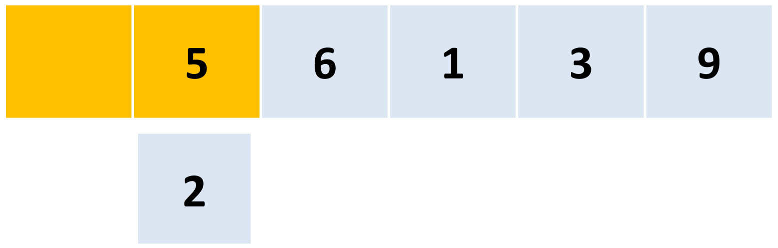
Step 1: assume first item is ”sorted”



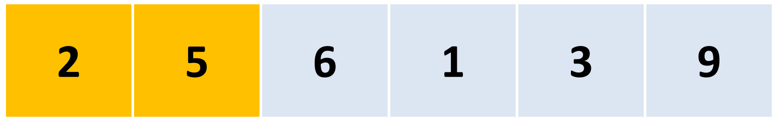
Step 2: remove the next item from array



Step 3: since 5>2, shift 5 to create space



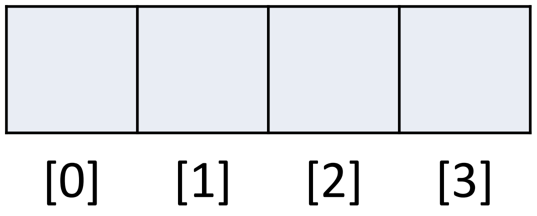
Step 4: insert 2 into the empty space

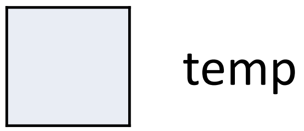


Exercise: implement a function that performs insertion sort in ascending order

This function takes one argument: a pointer to the array.

```
#define SIZE 4
void insertion_sort(int array[]){
   int i, j, temp, empty_idx;
   for(i=1;i<__________;i++){
      /* start with the 2nd element in the array */
      temp = array[i];
      empty_idx = __________;
      for(j=i-1;j>__________;j--){
         if(temp < array[j]){
            /* shift element to the right */

            /* update empty_idx */

         }
      }/* end of inner for loop */
      /* insert temp at empty_idx */
      array[_______________] = temp;
   }/* end of outer for loop */
}
```

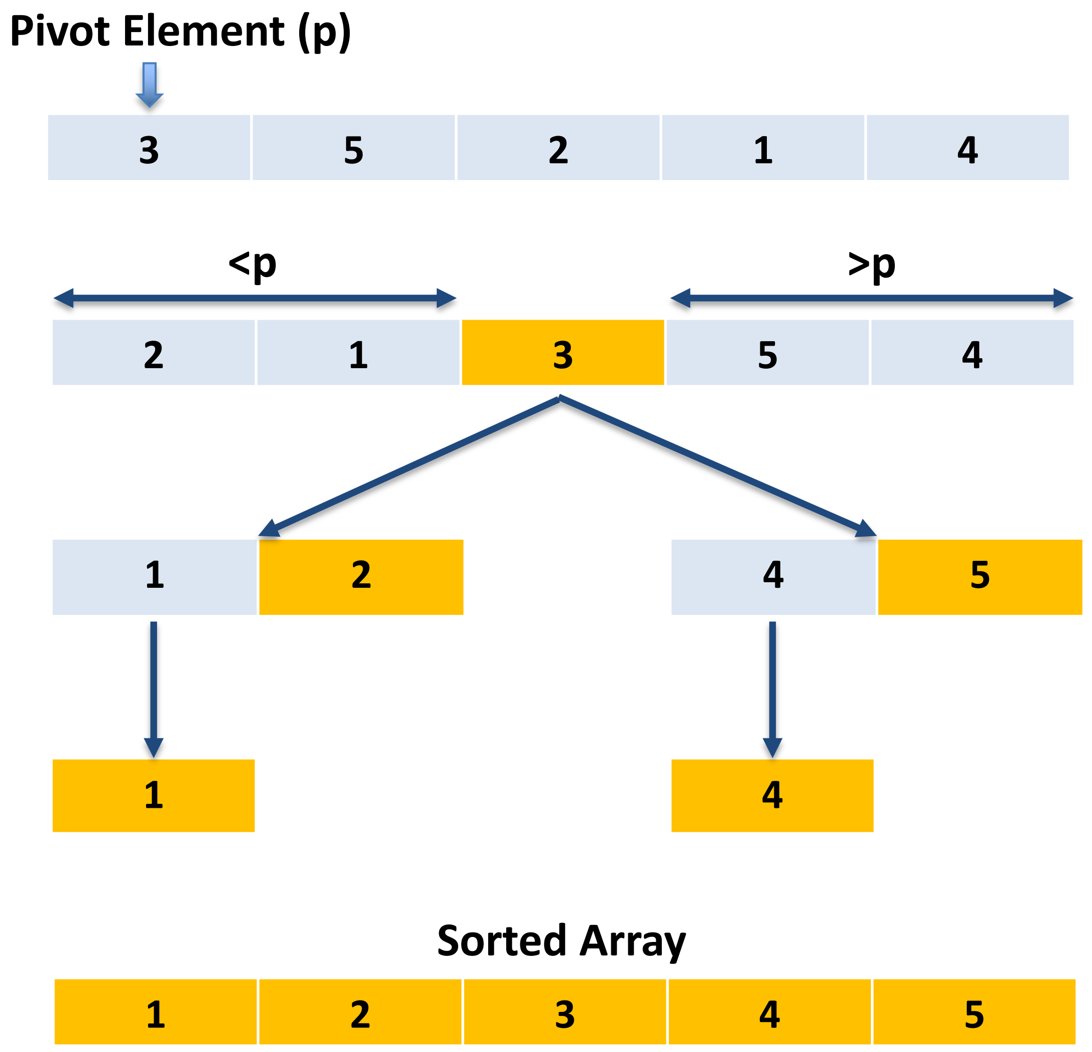




[ ] empty_idx

Quick Sort: also called divide-and-conquer

1. pick a pivot and partition array into 2 subarrays;
2. then sort subarrays using the same method.



Recursion

A **recursive function** is one that solves its task by **calling itself** on smaller pieces of data.

- Similar to recurrence function in mathematics
- Like iteration -- can be used interchangeably;
  sometimes recursion results in a simpler solution
- Must have at least 1 **base case** (terminal case) that ends the recursive process

Example: Running Sum ($\sum_1^n i$)

Mathematical Definition:

```
RunningSum(1) = 1
RunningSum(n) =
  n + RunningSum(n-1)
```

Recursive Function:

```
int RunningSum(int n) {
  if (n == 1)
    return 1;
  else
    return n + RunningSum(n-1);
}
```

Fibonacci Number

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

$$\begin{cases} F_n = F_{n-1} + F_{n-2} \\ F_0 = 0 \\ F_1 = 1 \end{cases}$$

```
int Fibonacci(int n){ /* assume n is non-negative */









}
```

Fibonacci with Look-up Table

```
int table[100];
/* assume each element will be initialized to -1 in main */

int fibonacci(int n){ /* assume 0<=n<100 */
   /* if fibonacci(n) has been calculated, return it */



   /* otherwise, perform the calculation, save it to table
     and then return it */









}
```

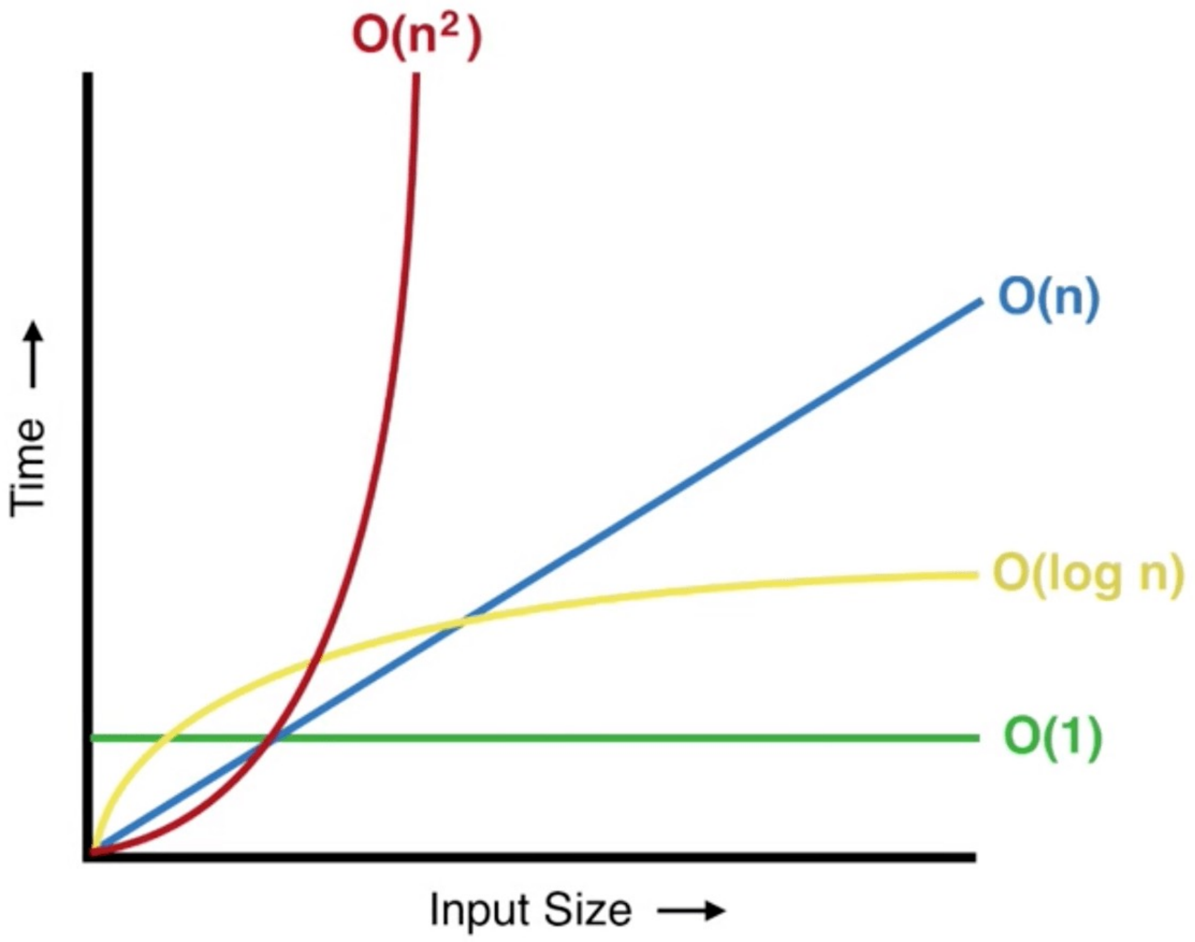
Recursive Binary Search

```
/* This function takes four arguments: pointer to a sorted array
in ascending order, the search item, the start index and the end
index of the array. If the search item is found, the function
returns its index in the array. Otherwise, it returns -1. */

int binary(int array[], int item, int start, int end){











}
```

Quick Sort

```
/* Assume partition() function is given. It chooses a pivot and
returns the index of the pivot after partitioning the array. */
int partition(int array[], int start, int end);

/* This function takes 3 arguments: a pointer to the array, the
start index of the array and the end index of the array. The array
should be sorted in ascending order after the function call. */

void quicksort(int array[], int start, int end){











}
```

A Sneak Peek at Big-O



https://levelup.gitconnected.com/big-o-time-complexity-what-it-is-and-why-it-matters-for-your-code-6c08dd97ad59