Lecture 2

- You have reviewed merge sort from the video
- This lecture, we will cover:
  - How to analyze the run time for recursive algorithms
  - How to prove correctness for recursive algorithms using proof by induction

Merge Sort

```
Algo **MergeSort** (A)
    n=length(A)
    If n>1
        Left=A[0,…,n/2-1]
        Right=A[n/2,…, n-1]
        Left=MergeSort(Left)
        Right=MergeSort(Right)
        return(Merge(Left,Right))
    Return A
```

How do we figure out the runtime when we have recursions in the algorithm?

Short answer: we use recursion too!

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n + c$$

```
Algo **Merge** (A,B)
    i,j,k=0, n=length(A), m=length(B)
    merged = array(n+m)
    while i≤ n & j≤ $m$
        if A[i]<B[j]
            merged[k]=A[i]
            i=i+1
        else
            merged[k]=B[j]
            j=j+1
        k=k+1
    if i< n merged[k:end]=A[i:n-1]
    if j< m merged[k: end]=B[j:m-1]
    return merged
```

Solve recurrence relation using telescoping

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n + c$$

Solve recurrence relation using recursion tree

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n + c$$

What if we break the array into 3 parts?

- Recurrence relation?
- Recursion tree?