CS 32 Week 7
Discussion 2E

Srinath

Outline

- Algorithm Analysis
- Sorting
- Worksheet 7 | Project 3 OH

Algorithm Analysis

Algo Analysis : Notations

A way to measure performance of an algorithm. Helps compare various algorithms performing the same task.

Big-O Notation

A function **f(N)** is **O(g(N)),** if there exists **N_0** and **k** such that **for all N >= N_0, |f(N)| <= k*g(N)**

3N^2-4N+2

Order -

Algo Analysis : Notations

A way to measure performance of an algorithm. Helps compare various algorithms performing the same task.

Big-O Notation
A function **f(N)** is **O(g(N)),** if there exists **N_0** and **k** such that **for all N >= N_0, |f(N)| <= k*g(N)**

3N^2-4N+2
Order - O(N^2)
Can we also write O(N^3) ? -

Algo Analysis : Notations

A way to measure performance of an algorithm. Helps compare various algorithms performing the same task.

Big-O Notation
A function **f(N)** is **O(g(N)),** if there exists **N_0** and **k** such that **for all N >= N_0, |f(N)| <= k*g(N)**

3N^2-4N+2
Order - O(N^2)
Can we also write O(N^3) ? - Yes, but generally report best possible bound

For algorithms, we try to measure the number of **basic steps** it performs for input size of **N.**

Basic steps : assignment, addition, subtraction etc.

Algo Analysis : Examples 1

```
int i, j;
for(i=0; i<10; i++){
        j = j+1;
}
```

Algo Analysis : Examples 1

```
int i, j;
for(i=0; i<10; i++){
        j = j+1;
}
```

O(1)

```
int i, j;
for(i=0; i<100000; i++){
        j = j+1;
}
```

Algo Analysis : Examples 1

```
int i, j;
for(i=0; i<10; i++){
        j = j+1;
}
```

O(1)

```
int i, j;
for(i=0; i<100000; i++){
        j = j+1;
}
```

O(1)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i++){
        j = j+1;
}
```

Algo Analysis : Examples 1

```
int i, j;
for(i=0; i<10; i++){
        j = j+1;
}
```

O(1)

```
int i, j;
for(i=0; i<100000; i++){
        j = j+1;
}
```

O(1)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i++){
        j = j+1;
}
```

O(N)

```
int i, j;
for(i=0; i<100; i++){
        for(j=0; j<100; j++){
                cout<<”hello”<<endl;
        }
}
```

Algo Analysis : Examples 1

```
int i, j;
for(i=0; i<10; i++){
        j = j+1;
}
```

O(1)

```
int i, j;
for(i=0; i<100000; i++){
        j = j+1;
}
```

O(1)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i++){
        j = j+1;
}
```

O(N)

```
int i, j;
for(i=0; i<100; i++){
        for(j=0; j<100; j++){
                cout<<”hello”<<endl;
        }
}
```

O(1)

Algo Analysis : Examples 2

```
int i, j, M, N;
cin>>M; cin>>N;
for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            cout<<”bruins”<<endl;
        }
}
```

Algo Analysis : Examples 2

```
int i, j, M, N;
cin>>M; cin>>N;
for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            cout<<”bruins”<<endl;
        }
}
```

O(M*N)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i=i+3){
        j = j+1
}
```

Algo Analysis : Examples 2

```
int i, j, M, N;
cin>>M; cin>>N;
for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            cout<<"bruins"<<endl;
        }
}
```

O(M*N)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i++){
        for(j=0; j<i; j++){
                cout<< "bruins" <<endl;
        }
}
```

```
int i, j, N;
cin>>N;
for(i=0; i<N; i=i+3){
        j = j+1
}
```

O(N)

Algo Analysis : Examples 2

```
int i, j, M, N;
cin>>M; cin>>N;
for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            cout<<"bruins"<<endl;
        }
}
```

O(M*N)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i++){
        for(j=0; j<i; j++){
                cout<< "bruins" <<endl;
        }
}
```

O(N^2)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i=i+3){
        j = j+1
}
```

O(N)

```
int i, j, N;
cin>>N;
for(i=10; i<N/2; i++){
        for(j=5; j<N/2; j++){
                cout<< "bruins" <<endl;
        }
}
```

Algo Analysis : Examples 2

```
int i, j, M, N;
cin>>M; cin>>N;
for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            cout<<"bruins"<<endl;
        }
}
```

O(M*N)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i++){
        for(j=0; j<i; j++){
                cout<< "bruins" <<endl;
        }
}
```

O(N^2)

```
int i, j, N;
cin>>N;
for(i=0; i<N; i=i+3){
        j = j+1
}
```

O(N)

```
int i, j, N;
cin>>N;
for(i=10; i<N/2; i++){
        for(j=5; j<N/2; j++){
                cout<< "bruins" <<endl;
        }
}
```

O(N^2)

Algo Analysis : More Examples

```
int i, j, N;
cin>>N;
for(i=0; i<N; i = 2*i){
        j = j+1;
}
```

Algo Analysis : More Examples

```
int i, j, N;
cin>>N;
for(i=0; i<N; i = 2*i){
        j = j+1;
}
```

This runs forever..

```
int i, j, N;
cin>>N;
for(i=1; i<N; i = 2*i){
        j = j+1;
}
```

Algo Analysis : More Examples

```
int i, j, N;
cin>>N;
for(i=0; i<N; i = 2*i){
        j = j+1;
}
```

This runs forever..

```
int i, j, N;
cin>>N;
for(i=1; i<N; i = 2*i){
        j = j+1;
}
```

O(log N)

```
int i, j, N;
cin>>N;
for(i=1; i<N; i++){
        j = isPrime(i);
}
```

Assume **isPrime(m)** is of **O(m^2)**

Algo Analysis : More Examples

```
int i, j, N;
cin>>N;
for(i=0; i<N; i = 2*i){
        j = j+1;
}
```

This runs forever..

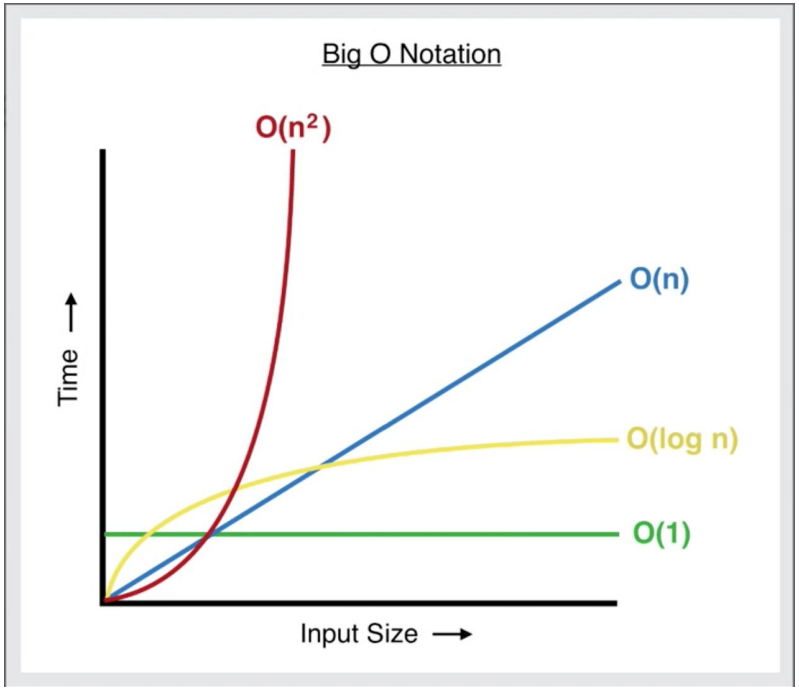
```
int i, j, N;
cin>>N;
for(i=1; i<N; i = 2*i){
        j = j+1;
}
```

O(log N)

```
int i, j, N;
cin>>N;
for(i=1; i<N; i++){
        j = isPrime(i);
}
```

Assume **isPrime(m)** is of **O(m^2)**
O(N^3)

Algo Analysis : Growth functions



Sorting

Sorting : Bubble Sort

		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	**5**			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	**4**				
	1	1	2	3					
i = 6	0	1	2	**3**					
		1	**2**						

Sorting : Bubble Sort

		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i=1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i=2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i=3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i=4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i=5	0	1	2	3	4				
	1	1	2	3					
i=6	0	1	2	3					
		1	2						

Time Complexity :
Space Complexity :

Sorting : Bubble Sort

		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i=1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i=2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i=3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i=4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i=5	0	1	2	3	4				
	1	1	2	3					
i=6	0	1	2	3					
		1	2						

Time Complexity : O(N^2)
Space Complexity : O(1)

Sorting : Selection Sort

```
void selectionSort(int arr[], int n)
{
}
```

Sorting : Selection Sort

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++){
        if (arr[j] < arr[min_idx])
            min_idx = j;
       }
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Sorting : Selection Sort

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++){
        if (arr[j] < arr[min_idx])
            min_idx = j;
       }
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Time Complexity :
Space Complexity :

Sorting : Selection Sort

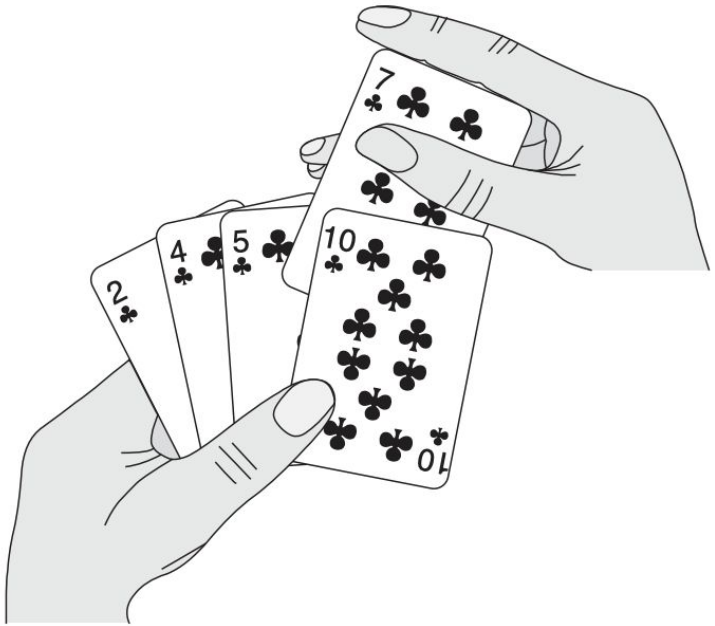
```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++){
        if (arr[j] < arr[min_idx])
            min_idx = j;
       }
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Time Complexity : O(N^2)
Space Complexity : O(1)

Sorting : Insertion Sort

6, 8, 21, 3, 9

Assume sorting a set of playing cards.



Sorting : Insertion Sort

6, 8, 21, 3, 9

6, 8, 21, 3, 9

6, 8, 21, 3, 9

6, 8, 21, 3, 9

6, 8, 21, 3, 9

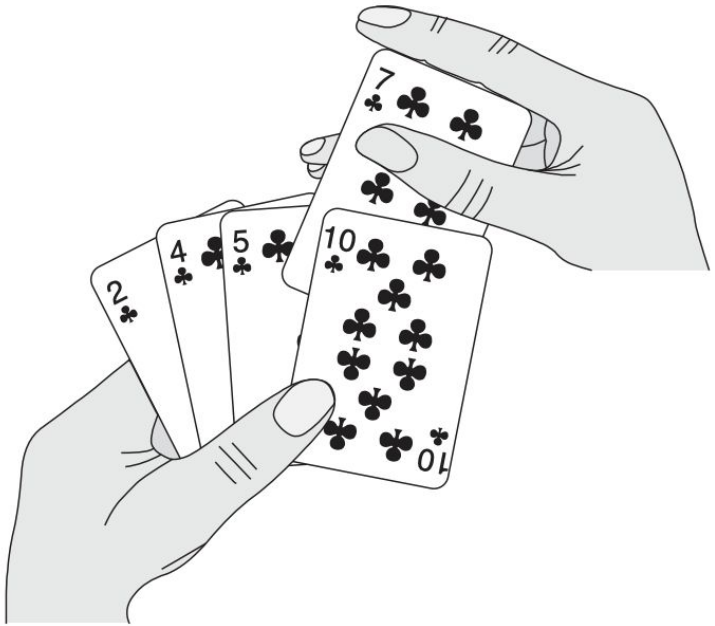
3, 6, 8, 21, 9

3, 6, 8, 21, 9

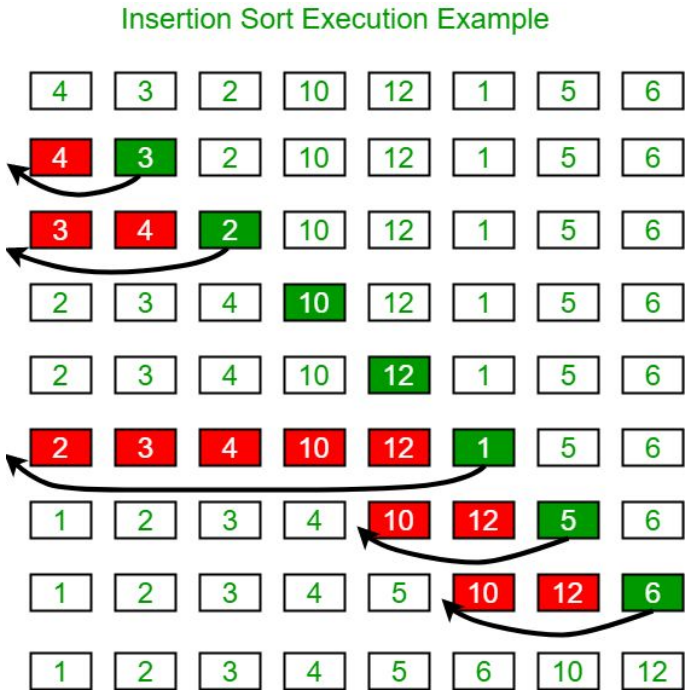
3, 6, 8, 9, 21

3, 6, 8, 9, 21

Assume sorting a set of playing cards.



Sorting : Insertion Sort



Sorting : Insertion Sort

```
void insertion_sort(int * A, int N){
        if(A == NULL || N <= 1) return;
        for(int j=1; j<N; j++){
                for(int i=j-1; i>=0; i--){
                        if(A[i] < A[i+1]) break;
                        else{
                                int tmp = A[i];
                                A[i] = A[i+1];
                                A[i+1] = tmp;
                                }
                }
        }
}
```

Time Complexity :
Space Complexity :

Sorting : Insertion Sort

```
void insertion_sort(int * A, int N){
        if(A == NULL || N <= 1) return;
        for(int j=1; j<N; j++){
                for(int i=j-1; i>=0; i--){
                        if(A[i] < A[i+1]) break;
                        else{
                                int tmp = A[i];
                                    A[i] = A[i+1];
                                A[i+1] = tmp;
                                }
                }
        }
}
```

Time Complexity : O(N^2)
Space Complexity : O(1)

Sorting : Merge-Sort

Given two sorted arrays, seq1, seq2 How can we get a single sorted array containing all the elements?

Seq1 : 2, 4, 6, 8, 9, 11, 18, 20
Seq2 : 1, 3, 5, 7, 10, 15
Sorted Array : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 18, 20

Sorting : Merge-Sort

Given two sorted arrays, seq1, seq2 How can we get a single sorted array containing all the elements?

Seq1 : 2, 4, 6, 8, 9, 11, 18, 20
Seq2 : 1, 3, 5, 7, 10, 15
Sorted Array : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 18, 20

```
void MERGE(int * A, int p, int q, int r) {
        int n1 = q-p+1;
        int n2 = r-q; // not considering q
        int L[n1+1];
        int R[n2+1];

        for(int i=0; i<n1; i++){
                L[i] = A[p+i];
        }
        L[n1] = INT_MAX;

        for(int i=0; i<n2; i++){
                R[i] = A[q+i+1];
        }
        R[n2] = INT_MAX;

        int i = 0;
        int j = 0;
        for(int k=p; k<=r ; k++){
                if(L[i] <= R[j]){
                        A[k] = L[i];
                        i = i+1;
                }else {
                        A[k] = R[j];
                        j = j+1;
                }
        }
        return;
}
```

The Merge Step

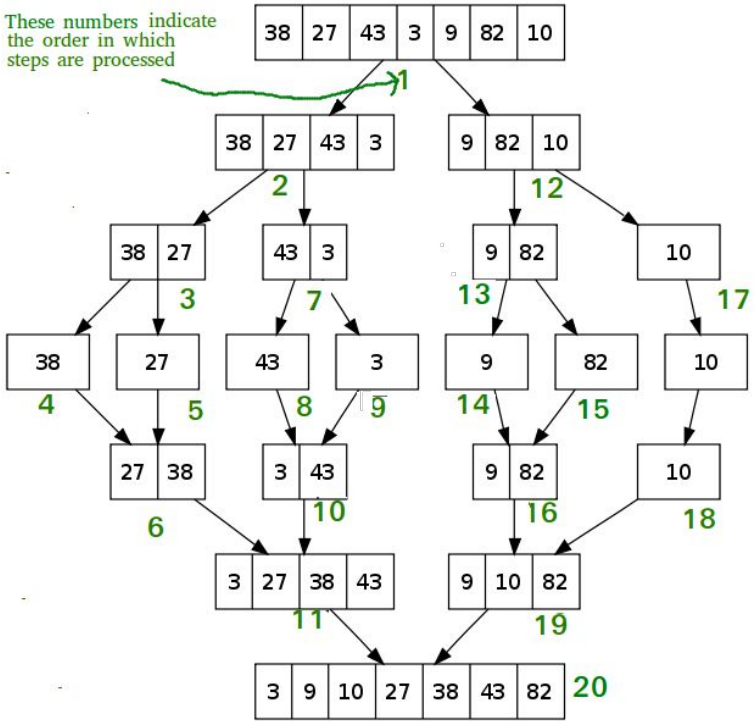
Sorting : Merge-Sort

Given two sorted arrays, seq1, seq2 How can we get a single sorted array containing all the elements?

Seq1 : 2, 4, 6, 8, 9, 11, 18, 20
Seq2 : 1, 3, 5, 7, 10, 15
Sorted Array : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 18, 20

Merge Sort

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```



Sorting : Merge-Sort

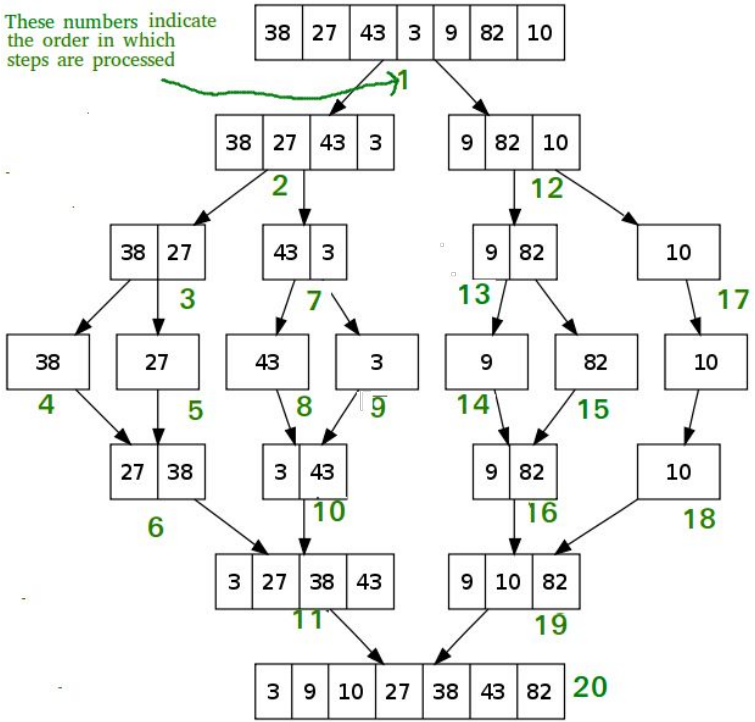
Given two sorted arrays, seq1, seq2 How can we get a single sorted array containing all the elements?

Seq1 : 2, 4, 6, 8, 9, 11, 18, 20
Seq2 : 1, 3, 5, 7, 10, 15
Sorted Array : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 18, 20

Merge Sort

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

Time Complexity :
Space Complexity :



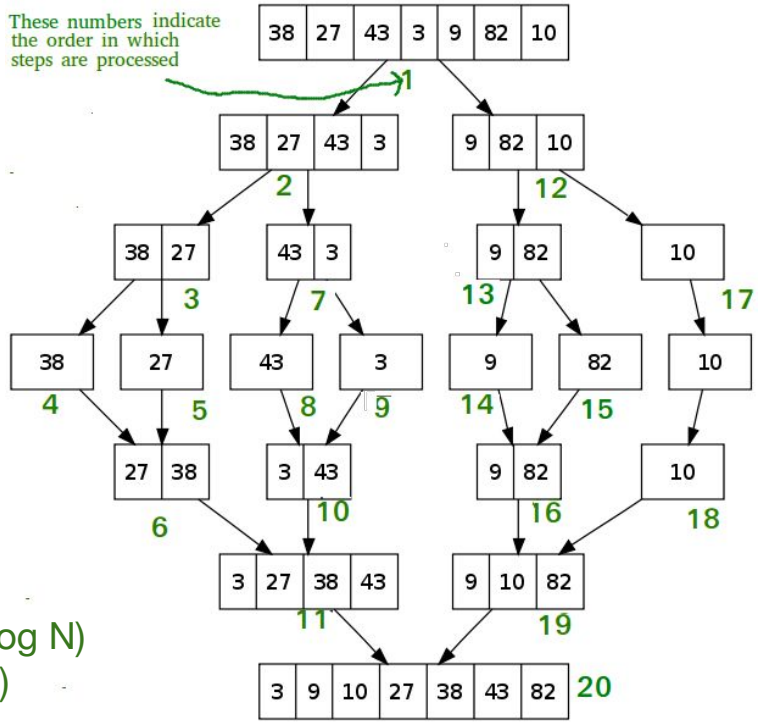
Sorting : Merge-Sort

Given two sorted arrays, seq1, seq2 How can we get a single sorted array containing all the elements?

Seq1 : 2, 4, 6, 8, 9, 11, 18, 20
Seq2 : 1, 3, 5, 7, 10, 15
Sorted Array : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 18, 20

Merge Sort

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```



Time Complexity : O(N log N)
Space Complexity : O(N)

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

Sorting : Merge-Sort

Time Complexity : O(N log N)
How did we get this?

T(N) = 2*T(N/2)+O(N), T(1) = 1
Can you solve this recursion ?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

Sorting : Merge-Sort

Time Complexity : O(N log N)
How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1
Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1

Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)

= 4*T(N/4)+2*O(N/2)+O(N)

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1

Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)
= 4*T(N/4)+2*O(N/2)+O(N)
= 4*(2*T(N/8)+O(N/4))+2*O(N/2)+O(N)

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1

Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)
= 4*T(N/4)+2*O(N/2)+O(N)
= 4*(2*T(N/8)+O(N/4))+2*O(N/2)+O(N)
= 8*T(N/8)+4*O(N/4))+2*O(N/2)+O(N)

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1

Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)
= 4*T(N/4)+2*O(N/2)+O(N)
= 4*(2*T(N/8)+O(N/4))+2*O(N/2)+O(N)
= 8*T(N/8)+4*O(N/4))+2*O(N/2)+O(N)
= …….
= …….
= 2^k * T(N/2^k) + k*O(N)

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1

Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)
= 4*T(N/4)+2*O(N/2)+O(N)
= 4*(2*T(N/8)+O(N/4))+2*O(N/2)+O(N)
= 8*T(N/8)+4*O(N/4))+2*O(N/2)+O(N)
= …….
= …….
= 2^k * T(N/2^k) + k*O(N)

Using base case, Set N/2^k = 1

Sorting : Merge-Sort

Time Complexity : O(N log N)

How did we get this?

```
void MERGE_SORT(int * A, int p, int r) {
        if(p < r){
                int q = (p+r)/2;
                MERGE_SORT(A, p, q);
                MERGE_SORT(A, q+1, r);
                MERGE(A, p, q, r);
        }
}
```

T(N) = 2*T(N/2)+O(N), T(1) = 1

Can you solve this recursion ?

T(N) = 2*(2*T(N/4)+O(N/2)) + O(N)
= 4*T(N/4)+2*O(N/2)+O(N)
= 4*(2*T(N/8)+O(N/4))+2*O(N/2)+O(N)
= 8*T(N/8)+4*O(N/4))+2*O(N/2)+O(N)
= …….
= …….
= 2^k * T(N/2^k) + k*O(N)

Using base case, Set N/2^k = 1
⇒ N = 2^k
⇒ k = log N
⇒ T(N) = 2^k*T(1) + k*O(N)
⇒ T(N) = N*1 + log N * O(N)

Therefore, T(N) ~ O(N log N)

Sorting : Quick Sort

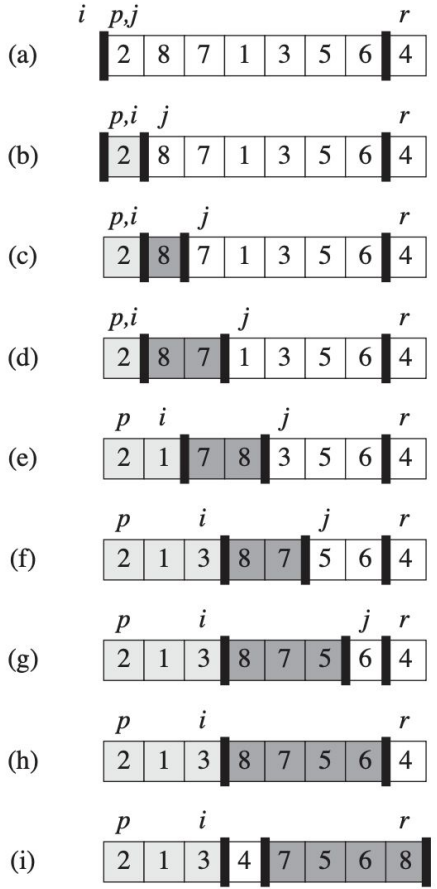
Given an array **A** of **n** elements, rearrange(partition) the array such that **A[n-1]** is at **q** th position and
A[i]<=A[q] for **0<=i<=q** and
A[i]>A[q] for **q<i<n**

Array : 2, 8, 7, 1, 3, 5, 6, 4
Re-arranged : 2, 1, 3, 4, 7, 5, 6, 8
There might be other arrangements which satisfy above conditions.

Sorting : Quick Sort

Given an array **A** of **n** elements, rearrange(partition) the array such that **A[n-1]** is at **q** th position and
A[i]<=A[q] for **0<=i<=q** and
A[i]>A[q] for **q<i<n**

Array : 2, 8, 7, 1, 3, 5, 6, 4
Re-arranged : 2, 1, 3, 4, 7, 5, 6, 8
There might be other arrangements which satisfy above conditions.



Sorting : Quick Sort

Given an array **A** of **n** elements, rearrange(partition) the array such that **A[n-1]** is at **q** th position and
A[i]<=A[q] for **0<=i<=q** and
A[i]>A[q] for **q<i<n**

Array : 2, 8, 7, 1, 3, 5, 6, 4
Re-arranged : 2, 1, 3, 4, 7, 5, 6, 8
There might be other arrangements which satisfy above conditions.

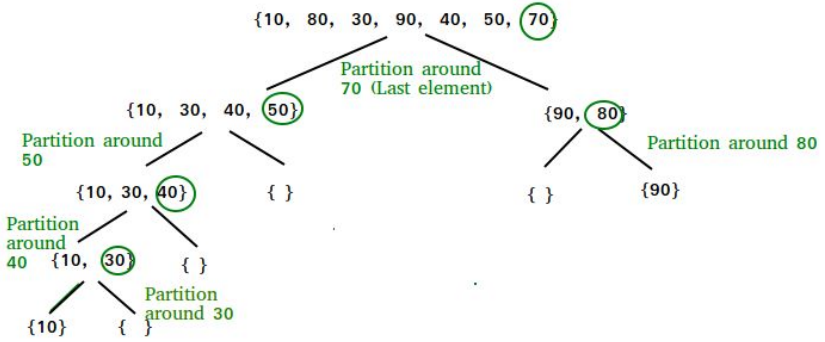
The Partition Step

```
void EXCHANGE(int * A, int i, int j){
        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
}

int PARTITION(int * A, int p, int r){
        int pivot = A[r];
        int i = p-1;
        for(int j=p ; j<r ; j++){
                if(A[j] <= pivot){
                        i = i+1;
                        EXCHANGE(A, i, j);
                }
        }
        EXCHANGE(A, i+1, r);
        return i+1;
}
```

Sorting : Quick Sort

Given an array **A** of **n** elements, rearrange(partition) the array such that **A[n-1]** is at **q** th position and
A[i]<=A[q] for **0<=i<=q** and
A[i]>A[q] for **q<i<n**

Array : 2, 8, 7, 1, 3, 5, 6, 4
Re-arranged : 2, 1, 3, 4, 7, 5, 6, 8
There might be other arrangements which satisfy above conditions.



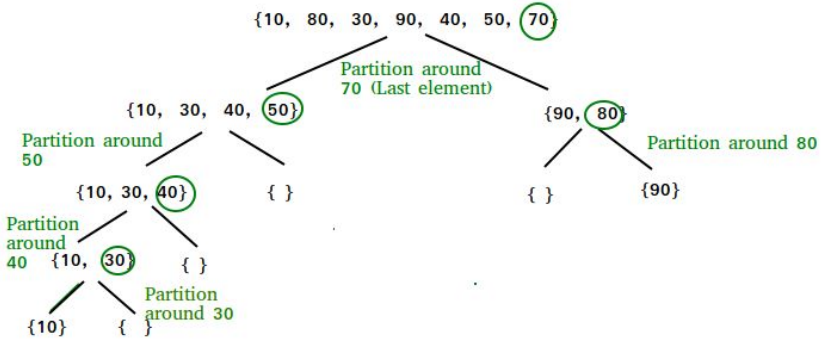
Quicksort

```
void QUICKSORT(int * A, int p, int r){
        if(p < r){
                int q = PARTITION(A, p, r);
                QUICKSORT(A, p, q-1);
                QUICKSORT(A, q+1, r);
        }
}
```

Sorting : Quick Sort

Given an array **A** of **n** elements, rearrange(partition) the array such that **A[n-1]** is at **q** th position and
A[i]<=A[q] for **0<=i<=q** and
A[i]>A[q] for **q<i<n**

Array : 2, 8, 7, 1, 3, 5, 6, 4
Re-arranged : 2, 1, 3, 4, 7, 5, 6, 8
There might be other arrangements which satisfy above conditions.



Quicksort

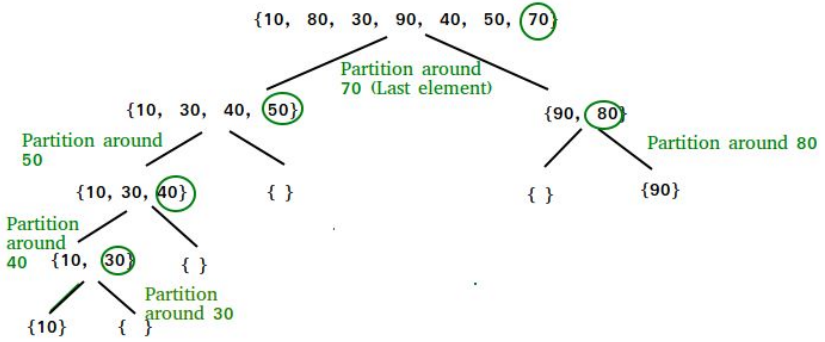
```
void QUICKSORT(int * A, int p, int r){
        if(p < r){
                int q = PARTITION(A, p, r);
                QUICKSORT(A, p, q-1);
                QUICKSORT(A, q+1, r);
        }
}
```

Time Complexity :
Space Complexity :

Sorting : Quick Sort

Given an array **A** of **n** elements, rearrange(partition) the array such that **A[n-1]** is at **q** th position and
A[i]<=A[q] for **0<=i<=q** and
A[i]>A[q] for **q<i<n**

Array : 2, 8, 7, 1, 3, 5, 6, 4
Re-arranged : 2, 1, 3, 4, 7, 5, 6, 8
There might be other arrangements which satisfy above conditions.



Quicksort

```
void QUICKSORT(int * A, int p, int r){
        if(p < r){
                int q = PARTITION(A, p, r);
                QUICKSORT(A, p, q-1);
                QUICKSORT(A, q+1, r);
        }
}
```

Time Complexity : O(N^2)-Worst case, O(N log N) - best/average case
Space Complexity : O(N)-Worst case, O(log N) - best/average case

Recursion

Recursion : Exercise

Count number of Nodes?

```
struct Node { ...; vector<Node*> children; };

int countNodes(Node* t){
    …..
 }
```

Recursion : Exercise

Count number of Nodes?

```
struct Node { ...; vector<Node*> children; };

int countNodes(Node* t){
        if (t == nullptr)
           return 0;
        int totalOfChildren = 0;
        for (int k = 0; k < t->children.size(); k++)
        {
            totalOfChildren += countNodes(t->children[k]);
        }
        return 1 + totalOfChildren;
 }
```

Recursion : Exercise

Count number of Nodes?

```
struct Node { ...; vector<Node*> children; };

int countNodes(Node* t){
        if (t == nullptr)
           return 0;
        int totalOfChildren = 0;
        for (int k = 0; k < t->children.size(); k++)
        {
            totalOfChildren += countNodes(t->children[k]);
        }
        return 1 + totalOfChildren;
}
```

Why the "+=" instead of "="? -

Recursion : Exercise

Count number of Nodes?

```
struct Node { ...; vector<Node*> children; };

int countNodes(Node* t){
        if (t == nullptr)
           return 0;
        int totalOfChildren = 0;
        for (int k = 0; k < t->children.size(); k++)
        {
            totalOfChildren += countNodes(t->children[k]);
        }
        return 1 + totalOfChildren;
}
```

Why the "+=" instead of "="? - We have to add all nodes of each children to get total count.
Why the "1 +"? -

Recursion : Exercise

Count number of Nodes?

```
struct Node { ...; vector<Node*> children; };

int countNodes(Node* t){
        if (t == nullptr)
           return 0;
        int totalOfChildren = 0;
        for (int k = 0; k < t->children.size(); k++)
        {
            totalOfChildren += countNodes(t->children[k]);
        }
        return 1 + totalOfChildren;
}
```

Why the "+=" instead of "="? - We have to add all nodes of each children to get total count.
Why the "1 +"? - Account for the **root** node on which the function is called.
Why isn't the return statement inside the loop? What would happen if it were? -

Recursion : Exercise

Count number of Nodes?

```
struct Node { ...; vector<Node*> children; };

int countNodes(Node* t){
        if (t == nullptr)
           return 0;
        int totalOfChildren = 0;
        for (int k = 0; k < t->children.size(); k++)
        {
            totalOfChildren += countNodes(t->children[k]);
        }
        return 1 + totalOfChildren;
}
```

Why the "+=" instead of "="? - We have to add all nodes of each children to get total count.
Why the "1 +"? - Account for the **root** node on which the function is called.
Why isn't the return statement inside the loop? What would happen if it were? - Will explore only in left most branch and return.

References

Algorithms and Data Structures (I usually refer these)

- **Introduction to Algorithms, T.H Cormen**
- https://web.cs.ucla.edu/~srinath/static/pdfs/DataStructures&Algorithms_Cormen.pdf
- **Algorithm Design, Eva Tardos**
- https://web.cs.ucla.edu/~srinath/static/pdfs/AlgorithmDesign_%20EvaTardos.pdf

Basic sorting implementations - https://web.cs.ucla.edu/~srinath/programming.html

Content in some of the slides is taken from Sonia Jaiswal.