-재귀 알고리즘

1>자신의 메소드를 호출

2>메소드 종료조건 필요

3> top-down 방식으로 호출

4>현재값에서 변화하여 호출하다가 종료조건 맞으면 가장 down에 있는 메소드 종료. 결과 리턴하면서 down에서 top 방향으로 이동

5> 그 다음 메소드 수행된 결과 리턴 – top 방향 이동...

6> top 만나면 종료

7> 반복문(bottom-up)방식과 비교

2번씩 호출- 중복 호출 여러번

- 정렬 알고리즘(효율적- 시간적+공간적(메모리-변수 얼마나 ..)

bubble sort	인접한 데이터와 크기 비교하여 교환 5*4 3 4 2 1 5 3 2 4 1 5 3 2 1 4 5-->x x x x 5 오름차순 정렬 마지막 데이터 3 2 1 4 2 3 1 4 2 1 3 4 --> x x x 4 5 정렬 2 1 3 .....
selection sort	아직 정렬되지 않은 데이터 중 1개 선택하여 정렬되지 않은 나머지 데이터들과 비교하여 교환 3 4 2 1 5 2 4 3 1 5 1 4 3 2 5--> 1 가장 작은 값 .. 1 2 .....
insertion sort	아직 정렬되지 않은 데이터를 정렬된 데이터들과 비교하여 해당 위치로 삽입 3 4 2 1 5 ==> 2 복사 ==> 0 3 4 1 5 ==>2보다 큰 데이터 밀어냄 ==> 2 3 4 1 5 ---> 앞쪽 삽입
shell sort	insertion sort 보완 1 3 4 5 6 7 - 2 이웃한 데이터를 삽입하는 것이 아니라 멀리 떨어진 데이터를 삽입

	1>먼저 정렬해야 할 리스트를 일정한 기준에 따라 분류 2>연속적이지 않은 여러 개의 부분 리스트를 생성 3>각 부분 리스트를 삽입 정렬을 이용하여 정렬 4>모든 부분 리스트가 정렬되면 다시 전체 리스트를 더 적은 개수의 부분 리스트로 만든 후에 알고리즘을 반복 위의 과정을 부분 리스트의 개수가 1이 될 때까지 반복
quick sort	재귀. 분할 정복 알고리즘 -기준점(pivot)을 설정해 해당 기준으로 분할 1> 데이터셋중에서 한 원소를 선택한다. 이렇게 고른 원소를 피벗(Pivot)이라고 한다. 3 5 6 7 1 --> 1 3 5 6 7 2>일반적으로는 pivot을 데이터셋 가장 앞 혹은 가장 뒤, 가운데 원소로 설정한다 3>Pivot을 기준으로 Pivot보다 '작은' 원소들은 Pivot의 앞(왼쪽)으로, '큰' 원소들은 Pivot의 뒤(오른쪽)으로 이동시킨다. 4> Pivot을 제외하고, Pivot의 왼쪽 부분과 오른쪽 부분으로 데이터셋을 "분할"한다. 5> 분할된 두 데이터셋에 각각 1-4 과정을 반복한다. 6> 데이터셋이 더이상 분할이 되지 않을 때 종료한다. pivot 중심 나누어진 2개 그룹의 크기 균일하지 않다 분할 상태에서 정렬 + 합침
merge sort	재귀. 분할 정복 알고리즘 2개 그룹 크기 균일하다 분할 1개 그룹될 때까지 + 합병 상태에서 정렬

counting sort	계수 정렬. 도수정렬
	{ 1 3 4 5 1 2 3 4 5 1 2 3 4 5}-> 1-5 사이 범위 한정. 동일값 포함
	정렬
	1-3 2- 2....
	1 1 1 2 2 3... 4... 5,,,
	int data[] = { 1 3 4 5 1 2 3 4 5 1 2 3 4 5};
	int count [] =new int[5+1];
	count[1] = 1 등장횟수
	
	도수분포표
	a[0] = 1;
	a[1] = 2;
	b[a[0]] = xxx;
	b[1] =xxx

int i = 5; --> 1번--> o(1)

for(int i = 0; i<=10..){-->10번 --> o(n)

 .for(){

        ......

   }

}

==> 메소드 수행 --> o(n+1) == o(n)

o(1) < o(logn) < o(n) < o(n^2)

n=10 5 2 ...

bubble sort

selection sort

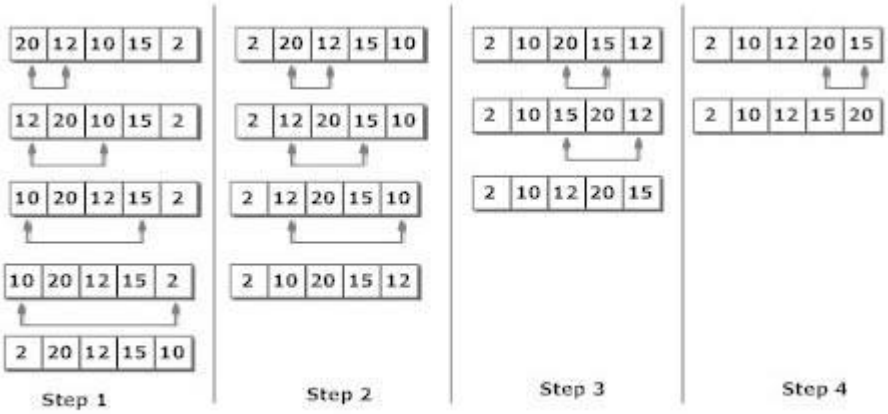
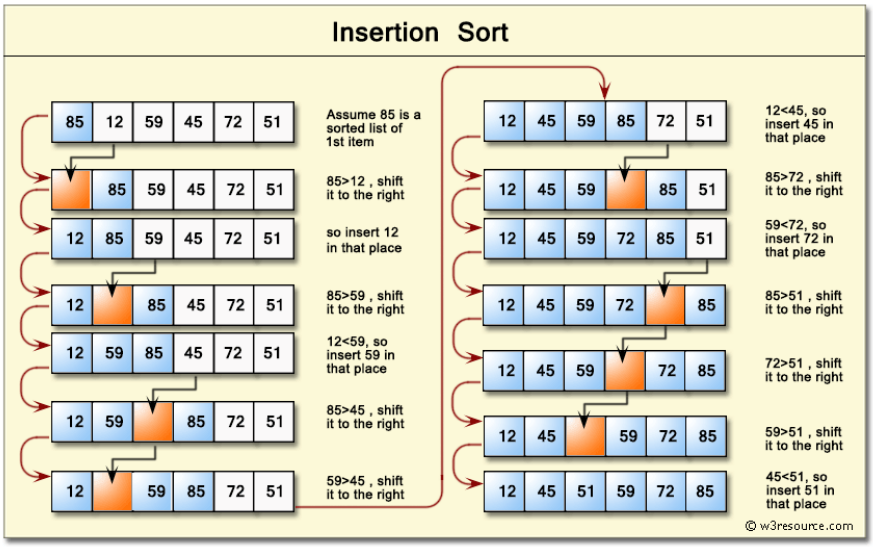
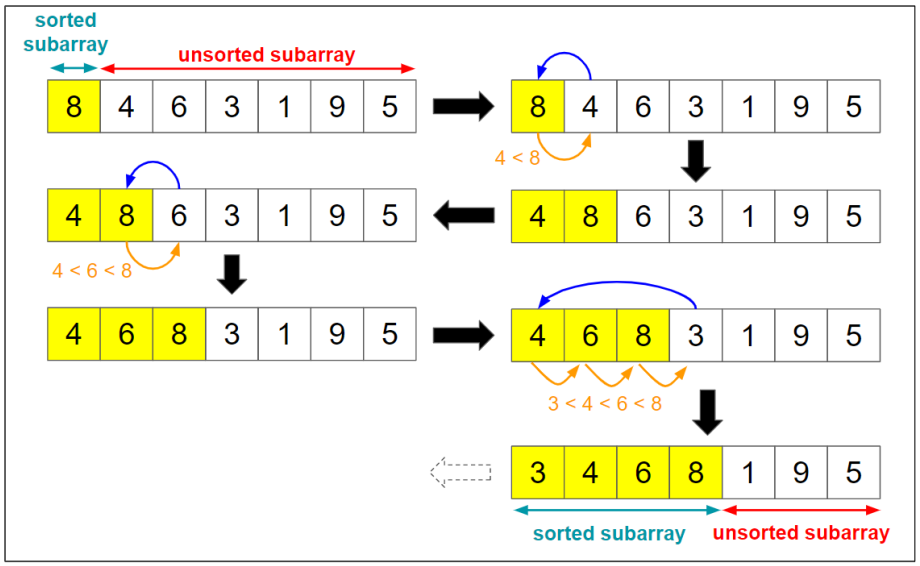


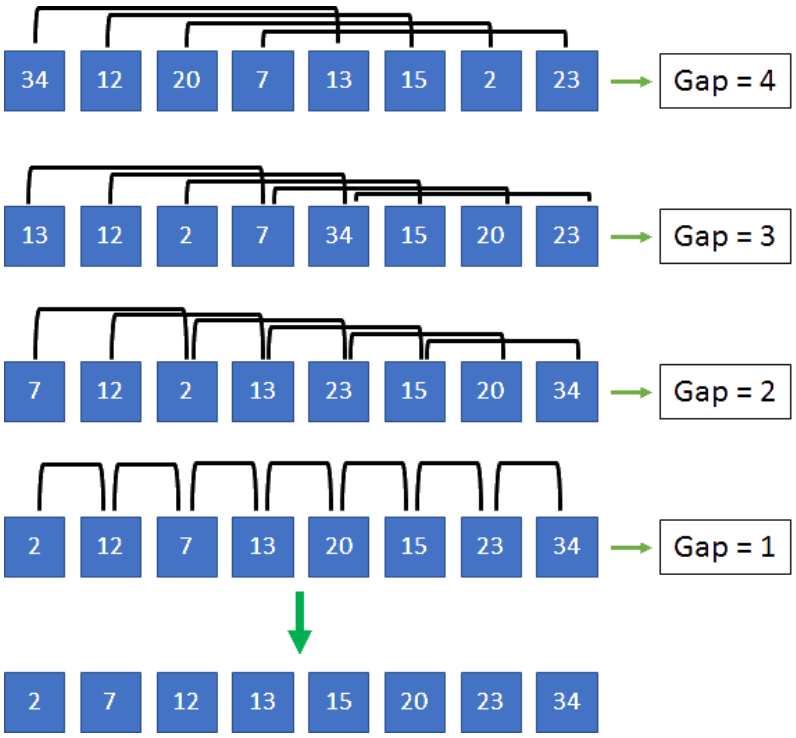
Figure: Selection Sort

insertion sort





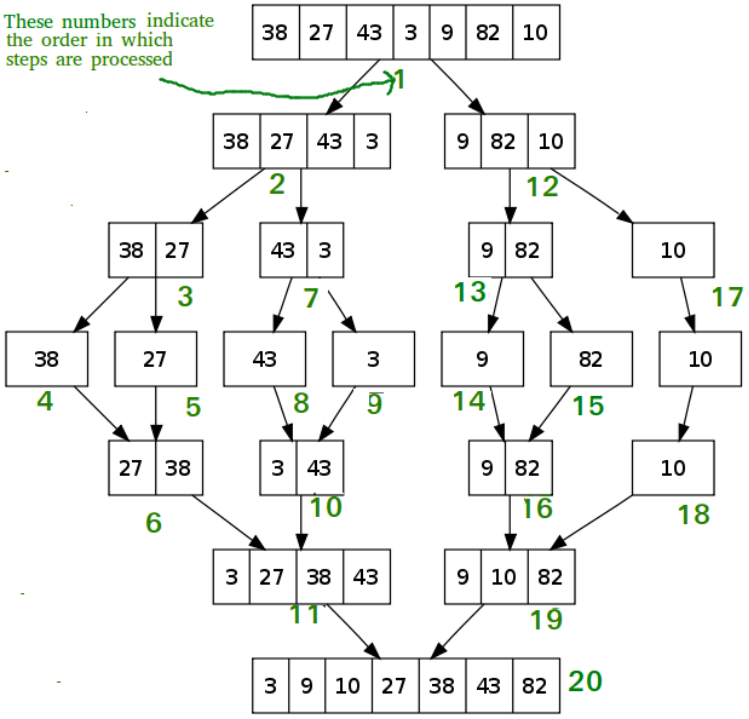
shell sort



- quick sort

```
{55 11 22 33 44 66 88}
피벗 33 : 데이터 88 비교하여 33 가 작음. pr 현재 6 에서 1 감소
피벗 33 : 데이터 66 비교하여 33 가 작음. pr 현재 5 에서 1 감소
피벗 33 : 데이터 44 비교하여 33 가 작음. pr 현재 4 에서 1 감소
```

```
pl=0 pr=3 위치에서 55 33데이터교환 발생
피벗 33 : 데이터 11 비교하여 33 가 큼 . pl 현재 1 에서 1 증가
피벗 33 : 데이터 22 비교하여 33 가 큼 . pl 현재 2 에서 1 증가
left=0 right=6 pl=3 pr=2
피벗 11 : 데이터 22 비교하여 11 가 작음. pr 현재 2 에서 1 감소
pl=0 pr=1 위치에서 33 11데이터교환 발생
left=0 right=2 pl=1 pr=0
pl=1 pr=2 위치에서 33 22데이터교환 발생
left=1 right=2 pl=2 pr=1
피벗 44 : 데이터 88 비교하여 44 가 작음. pr 현재 6 에서 1 감소
피벗 44 : 데이터 66 비교하여 44 가 작음. pr 현재 5 에서 1 감소
pl=3 pr=4 위치에서 55 44데이터교환 발생
left=3 right=6 pl=4 pr=3
피벗 66 : 데이터 55 비교하여 66 가 큼 . pl 현재 4 에서 1 증가
피벗 66 : 데이터 88 비교하여 66 가 작음. pr 현재 6 에서 1 감소
pl=5 pr=5 위치에서 66 66데이터교환 발생
left=4 right=6 pl=6 pr=4
```



6장 정렬

집합/리스트/문자열검색

10장 트리

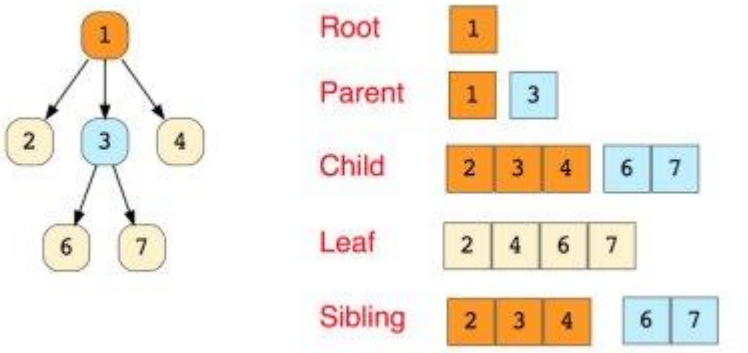
1> 부모 자식 연결 관계

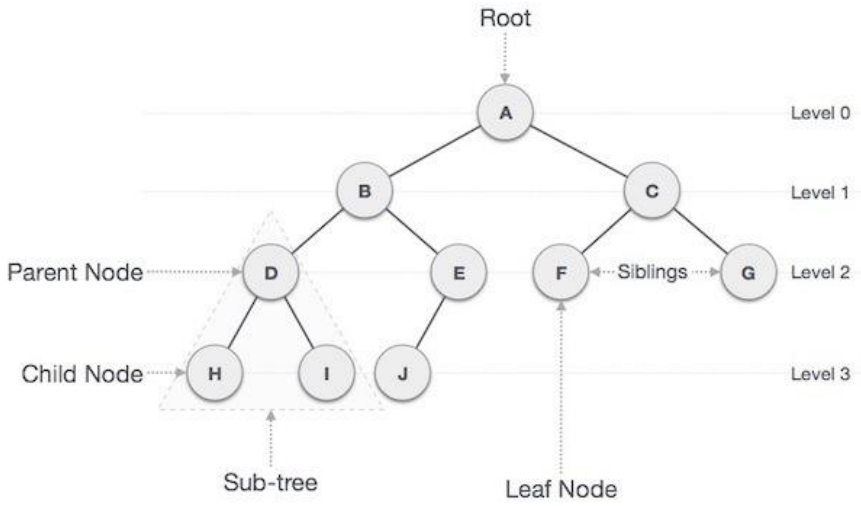
2> stack – LIFO

QUEUE - FIFO

TREE -

- 트리





1> 자바 트리 표현 클래스 = NODE 집합체

2> 이진트리 – 자식 0 -1 -2까지

```
 class Node {
    String name;
   Node left;
   Node right;
}
```

2> 트리 내부 모든 NODE 검색방문방법(이진트리)

2-1. BREADTH FIRST SEARCH (BFS)

2-2. DEPTH FIRST SEARCH(DFS)

2-2-1. 현재노드 –> 왼쪽자식 -> 오른쪽 자식

2-2-2. 왼쪽자식 -> 현재노드 –> 오른쪽 자식

2-2-2. 왼쪽자식–> 오른쪽 자식 -> 현재노드