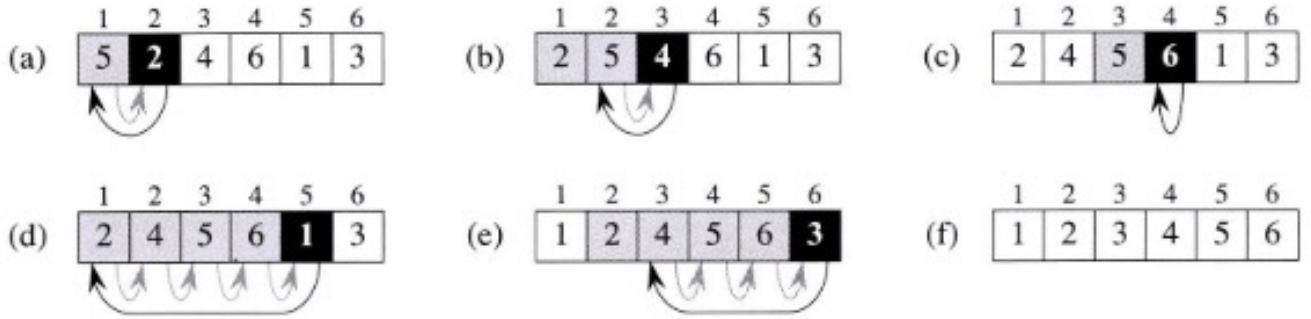
Pervasive Elastic MetaLearning Laboratory
Department of Computer Engineering
Hongik University

HW12 실습

Sorting

삽입 정렬 Insertion Sort

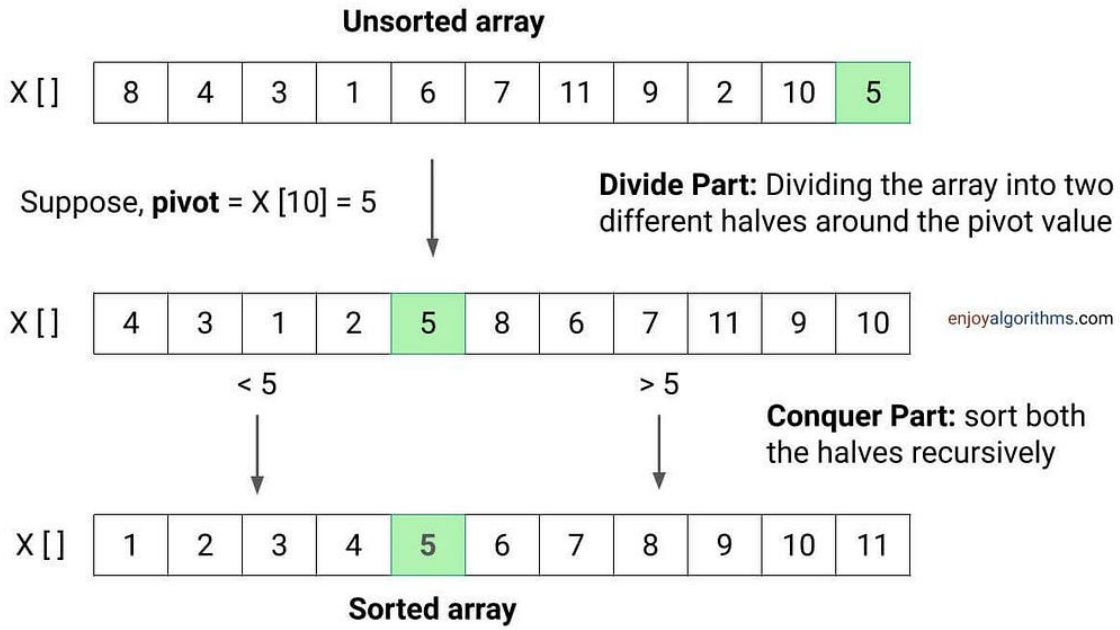
- 모든 요소를 앞에서부터 차례대로 이미 정렬된 배열 부분과 비교하여, 자신의 위치를 찾아 삽입으로써 정렬을 완성하는 알고리즘




```
template <class T>
void InsertionSort(T *a, int n)
{
  for (int j=2; j<=n; j++){
    T temp = a[j];
    Insert(temp, a, j-1);
  }
}
```

빠른 정렬 Quick Sort

1. 리스트 가운데서 하나의 원소(pivot)를 고른다
2. 피벗 앞에는 피벗보다 값이 작은 모든 원소들이 오고, 피벗 뒤에는 피벗보다 값이 큰 모든 원소들이 오도록 피벗을 기준으로 리스트를 둘로 나눈다
3. 분할된 두 개의 작은 리스트에 대해 재귀적으로 이 과정을 반복

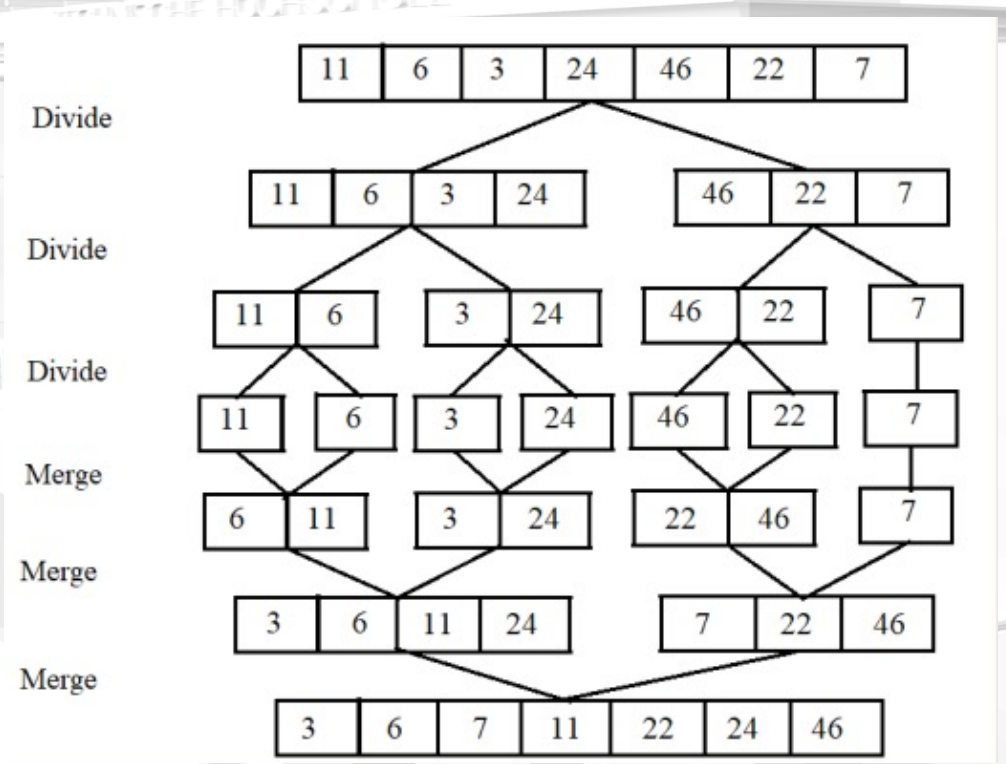



```
template <class T>
void QuickSort(T *a, const int left, const int right)
{
  if(left < right){
    int i = left, j = right+1, pivot = a[left];
    do{
      do i++; while(a[i] < pivot);
      do j--; while(a[j] > pivot);
      if(i < j) swap(a[i], a[j]);
      } while(i < j);
    swap(a[left],a[j]);

    QuickSort(a, left, j-1);
    QuickSort(a, j+1, right);
  }
}
```

합병 정렬 Merge Sort

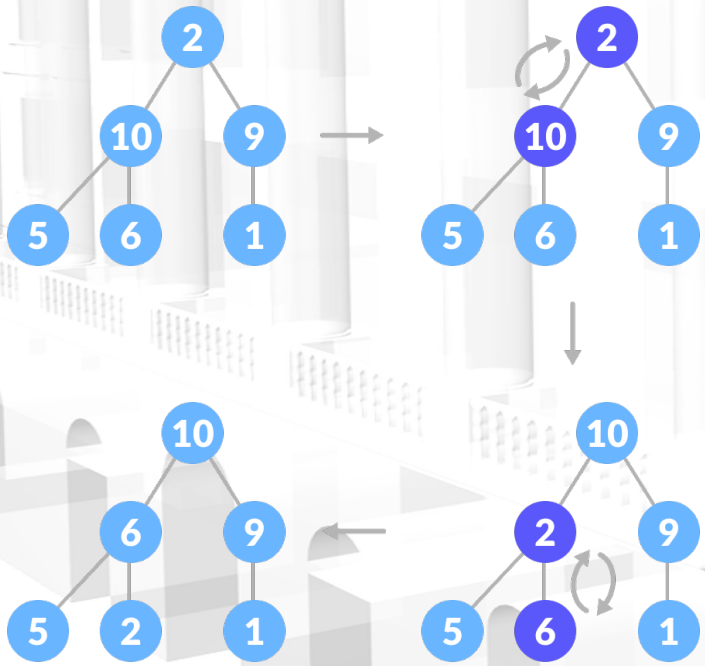
- 하나의 리스트를 두개의 균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 한다




```
template <class T>
void MergeSort(T *a, const int n)
{
  T *tempList = new T[n+1];
  for(int l=1; l < n; l *= 2)
  {
    MergePass(a, tempList, n, l);
    l *= 2;
    MergePass(tempList, a, n, l);
  }
  delete [] tempList;
}
```

힙 정렬 Heap Sort

- 최대 힙 트리나 최소 힙 트리를 구성해 정렬
- 내림차순 정렬 → 최소 힙
- 오름차순 정렬 → 최대 힙




```
template <class T>
void HeapSort(T *a, const int n) //힙으로 조정
{
  for(int i = n/2; i>=1; i--) Adjust(a,i,n);
  for(i=n-1; i>=1; i--)
  {
    swap(a[1],a[i+1]);
    Adjust(a,1,i); //힙으로 조정
  }
}
```

실습 내용

- (1) 삽입 정렬(Insertion Sort)
- (2) 빠른 정렬(Quick Sort)
- (3) 자연 합병 정렬(Natural Merge Sort)
- (4) 힙 정렬(Heap Sort)

- 위 알고리즘들을 구현하고
- 이들의 실행 시간을 측정하여 각 알고리즘의 복잡도를 분석

질문

- pemds81718@gmail.com
- 간단한 구글링으로 알 수 있는 내용은 답변하지 않습니다.