# Assignment #1
# Divide-and-Conquer Algorithms

인터넷미디어공학과
201312845 김일식

1. merge sort, maximum subarray algorithm 에 대해 (1) divide (2) conquer(특히 base case의 경우 어떻게 해결하는지) (3) combine 과정 정리, 서술.

merge sort - divide의 경우 초기 배열의 첫 인덱스, 끝 인덱스를 저장하고 그 중간값을 구한다. 그 후 중간값을 기준으로 왼쪽과 오른쪽을 나누어 각각의 배열을 새로 첫 인덱스, 끝 인덱스를 기준으로 잡고 다시 그 중간값을 구한다. 이 과정을 반복한다. 이때 계속 나누는 방법으로 recursive 함수의 방법을 사용한다. conquer의 경우 배열의 원소의 개수가 1개가 될 때 까지 나눈다.(base case) 1개는 이미 정렬된 상태이므로 함수를 리턴한다. 그 상위 배열은 왼쪽과 오른쪽을 가지고 있으므로, 이 왼쪽과 오른쪽을 모아서 다시 정렬한다. 이때 combine의 과정으로 진행한다. combine의 경우 왼쪽 배열과 오른쪽 배열의 원소들을 새로운 배열을 만들어 저장한다. 이 때 왼쪽과 오른쪽 배열은 정렬된 상태이다. 먼저 왼쪽 배열의 첫 번째 원소와 오른쪽 배열의 첫 번째 원소를 비교한다. 더 큰 것을 상위 배열의 첫 번째 원소로 넣는다. 그 후에 더 큰 원소를 가진 쪽의 배열은 그 원소 다음 원소를 기준으로 잡는다. 이렇게 첫 번째 원소부터 마지막 원소까지 차례대로 비교하여 작은 원소부터 상위 배열의 원소로 저장한다. 마지막까지 저장하면 combine 과정이 끝난다.

maximum subarray 알고리즘 - divide의 경우 merge sort와 마찬가지로 배열을 나눈다. 첫 인덱스와 끝 인덱스를 이용, 중간값을 구하고 왼쪽 배열과 오른쪽 배열을 각각 나누는 과정을 recursive 의 방법으로 반복한다. merge sort와 다른 점은 여기서 중간을 지나는 최대값을 구한다는 것이다. conquer의 경우 base case는 원소가 하나이다. 그렇다면 그 자체가 maximum이므로 리턴한다. 상위 배열부터는 왼쪽과 오른쪽 배열, 그리고 그 중간을 지나는 원소들 중에 최대값을 구한다. 왼쪽과 오른쪽 배열에서는 원소들의 합이 최대인 subarray를 구한다. 중간을 기준으로 하는 crossing subarray 로직에서는 중간을 기준으로 왼쪽과 오른쪽으로 각각 한 원소씩 추가하며 최대가 되는 값을 찾는다. 그렇게 구한 왼쪽, 오른쪽, 중간의 각각의 최대값 중 가장 큰 최대값을 상위 함수로 리턴한다. combine은 왼쪽과 오른쪽, 중간을 지나는 최대값 세 개 중 가장 큰 값을 찾아 상위 함수로 리턴하는 것의 경우라 할 수 있다.

2. Matrix-Multiply 알고리즘을 Divide and conquer 방식으로 고안.

n이 a power of 2이므로 반드시 행렬은 4개로 나누어진다. 즉, 예를 들어 4x4 행렬일 경우 2x2행렬 4개로 나눌 수 있다. 행렬 C가 A * B의 결과일 경우, C는 c11 ~ c44의 원소로 구성된다. 이때 4개의 행렬은 c11, c12, c21, c22의 행렬, c13, c14, c23, c24의 행렬, c31, c32, c41, c42의 행렬, c33, c34, c43, c44의 행렬로 나눌 수 있다. 이 생각에 착안하면 Divide and Conquer 방식은 n * n 행렬을 n / 2 * n / 2 행렬 4개로 나눌 수 있고, 이 것을 n = 1일 때 까지 할 수 있다. n = 1이라면 하나의 원소를 구하는 것이므로 결국 combine하여 행렬 전체를 만들 수 있다. 이것을 수도 코드로 나타내면 다음과 같다.

```
MatrixProduct(C, A, B, p, r, OriginalSize, CurSize)
1       i = 1;
2       if CurSize == 1
3               C_pr = sum_{i=1}^{OriginalSize} A_pi + B_ir
4       else
5               CurSize = CurSize / 2
6               MatrixProduct(C, A, B, p, r, OriginalSize, CurSize)
7               MatrixProduct(C, A, B, p + CurSize, r, OriginalSize, CurSize)
8               MatrixProduct(C, A, B, p, r + CurSize, OriginalSize, CurSize)
9               MatrixProduct(C, A, B, p + CurSize, r + CurSize, OriginalSize, CurSize)
```

여기서 p, r은 행렬의  가장 왼쪽 첫 번째 원소의 위치를 저장한다. 그것을 기점으로 다시 Divide할 수 있다.

3. Array A[]가 주어졌을 때 A의 element들 중에 max값과 min 값을 동시에 구하는 알고리즘을 Divide and Conquer 방식으로 고안.

먼저 배열을 첫 번째 인덱스와 마지막 인덱스의 중간 인덱스를 기준으로 Divide 한다. 나누어진 왼쪽 배열과 오른쪽 배열을 보았을 때 각각 원소가 하나가 남을 때 까지 Divide 한다. 원소가 하나가 남았을 때 그 것을 배열로 본다면 그 배열 안에서 MAX값과 MIN값은 동시에 그 원소의 값이 된다. 그렇게 상위 배열로 돌아가면 왼쪽과 오른쪽 의 각각의 MAX값과 MIN값을 비교하여 상위 배열의 MAX값과 MIN값을 찾아 Combine 한다. 이렇게 계속 상위 배열로 올라가면 결국 배열 전체에서 MAX값과 MIN 값을 찾게 된다. 이것을 수도 코드로 나타내면 다음과 같다.

```
FindMaxMin(Array A[], max, min, p, r)
1       LMin, LMax, RMin, RMax 선언
2       if p != r
3               q = (p + r) / 2
4               FindMaxMin(A, &LMin, &LMax, p, q)
5               FindMaxMin(A, &RMin, &RMax, q + 1, r)
6               if LMin < RMin
7                       min = LMin
8               else
9                       min = RMin
10              if LMax > RMax
11                      max = LMax
12              else
13                      max = RMax
14      else
15              max = min = A[p]
```

4. 평면 상의 n 개의 점이 주어졌을 때 가장 가까운 두 점을 찾는 알고리즘을 고안. 다음 두 조건을 만족하는 pi, pj를 찾아 리턴하는 Divide and Conquer 알고리즘을 작성.

(1) pi != pj

(2) 두 점사이의 거리가 최소인 pi, pj

각각의 점들을 담은 배열 A를 가정하자. 각 배열의 요소는 하나의 점이고 하나의 점은 두 원소로 구성된다. 즉, x,y로 구성된다. 이때 배열을 단순히 왼쪽과 오른쪽으로 나누면 안된다. 왜냐하면 왼쪽 배열에 속한 임의의 점과 오른쪽 배열에 속한 임의의 점의 거리가 최소일 수 있기 때문이다. 따라서 Maximum Subarray 알고리즘의 아이디어를 응용하여 왼쪽 배열 점들 사이의 최소 거리, 오른쪽 배열 점들 사이의 최소 거리, 중간을 지나며 만들어지는 배열 안에서의 점들 사이의 최소 거리를 구하여 무엇이 최소값인지를 비교하여야 한다. 끝까지 Divide하면 가장 가까운 두 점이 하나의 점을 가리키게 되므로 어떤식으로 해도 한 점을 가리키는 것이 가장 가깝다. 따라서 한 점을 가리키면 그 거리는 무한대로 두어야 오류를 방지할 수 있다. Conquer의 경우 가장 가까운 두 점을 찾아서 리턴한다. Combine단계에서는 왼쪽 배열, 오른쪽 배열, 중간을 지나는 배열의 가장 가까운 점들 중 거리가 최소인 점 두 개를 리턴한다. 배열이 나누어졌을 때 한 번만 두 점 사이의 거리를 구해 놓으면 상위 배열에서 거리만 비교하므로 다시 거리를 계산하지 않아도 된다. 이를 수도 코드로 나타내면 다음과 같다.

```
FindMinimumDistance(A[1… n], p, r, point1, point2)
1       d1 = ∞, d2 = ∞, d3 = ∞
2       if p == r
3               point1 = A[p]
4               point2 = A[p]
5               return ∞ (두 점 사이의 거리 최대)
6        else
7               q = (p + r) / 2
8               d1 = FindMinimumDistance(A, p, q)
9               d2 = FindMinimumDistance(A, q+1, r)
10              d3 = FindCrossingMinimumDistance(A, p, r, q)
11              if point1 != point2
12                      if d1 < d2 and d1 < d3
13                              return d1, point1, point2(왼쪽 배열에서 찾은 두 점)
14                      if d2 < d1 and d2 < d3
15                              return d2, point1, point2(오른쪽 배열에서 찾은 두 점)
16                      if d3 < d1 and d3 < d2
17                              return d3, point1, point2(중간 배열에서 찾은 두 점)
18              else // 점이 두개만 있는 배열일 때
19                      point1 = A[p]
20                      point2 = A[r]
21                      return distance(point1, point2)
```

중간을 지나는 배열에서 두 점 사이의 거리를 구하는 수도 코드는 다음과 같다.

```
FindCrossingMinimumDistance(A, p, r, q)
1       minDistance = ∞
2       for i = q downto p
3               for j = q + 1 to r
4                       if minDistance > distance(A[i], A[j])
5                               minDistance = distance(A[i], A[j])
6       return minDistance
```

5. Array A[1… n]가 n개의 element를 가지고 있다고 할 때 이 배열의 majority element는 n / 2번 보다 많이 존재하는 element를 의미한다. n = 6 또는 n = 7일 경우 3번 초과로 나오는 element. 단, 이 행렬의 element들은 서로 크기를 비교할 수 없다. 따라서 sort가 불가능하지만, 두 개의 element가 같은지는 check할 수 있다. 이때, A의 majority element를 찾아서 return하는 Divide and Conquer 알고리즘 고안.

배열의 원소가 1개일 때 까지 나눈다. 그때 majority element는 그 원소 1개가 되고 리턴된다. 상위 배열에서 원소가 2개 이상이라면 majority element는 있을 수도 있고 없을 수도 있다. 있을 때는 그 원소를 저장하고, 없을 때는 0을 리턴한다. 이때 모든 경우를 따져보면 다음과 같다. 양쪽 함수의 리턴값이 같다면 두 가지다. 둘 다 majority element가 없거나, 양 쪽의 majority element가 같은 경우다. 이때는 어떠한 경우든 그대로 상위 함수로 리턴하면 된다. 양쪽 함수의 리턴값이 다르다면 필시 한쪽은 majority element가 있다. 이때는 그 majority element가 현재 배열 전체의 majority element가 되는지를 검사해야 한다. 맞다면 그 원소를 리턴하고 아니면 0을 리턴한다. 이것을 수도 코드로 나타내면 다음과 같다.

```
FindMajorityElement(A, p, r)
1       if p == r
2               return A[p]
3       q = (p + r) / 2
4       lMaj = findMajElmt(A, p, q);
5       rMaj = findMajElmt(A, q + 1, r);
6       if lMaj = rMaj
7               return lMaj
8       else
9               if lMaj != 0
10                      n1 = getCount(A, p, r, lMaj)
11              if rMaj != 0
12                      n2 = getCount(A, p, r, rMaj)
13              if n1 >= n2 and n1 > (r - p) / 2
14                      return lMaj
15              else if n2 > n1 and n2 > (r - p) / 2
16                      return rMaj
17      return 0
```