ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

APPLIED ALGORITHMS

DIVIDE-AND-CONQUER

ONE LOVE. ONE FUTURE.

CONTENT

- Basis of Divide-And-Conquer
- Karatsuba algorithm
- Closest pair points
- Decrease and Conquer
- Inversion

Basis of Divide and Conquer

- Generic schema
  - Divide the original problem into smaller independent subproblems
  - Solve subproblems (recursion)
  - Combine solutions of subproblems

Basis of Divide and Conquer

- Complexity analysis
  - $T(n)$: running time of input size n

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq n_c \\ aT(n/b) + D(n) + C(n) & \text{if } n \geq n_c, \end{cases}$$

- Consider: $T(n) = aT(n/b) + n^k$ với a, b, c, k are positive constants and $a \geq 1, b \geq 2$:

→ $T(n) = \begin{cases} O\left(n^{\log_b a}\right), \text{ nếu } a > b^k \\ O\left(n^k \log n\right), \text{ nếu } a = b^k \\ O\left(n^k\right), \text{ nếu } a < b^k \end{cases}$

Multiplication of 2 big numbers: Karatsuba algorithm

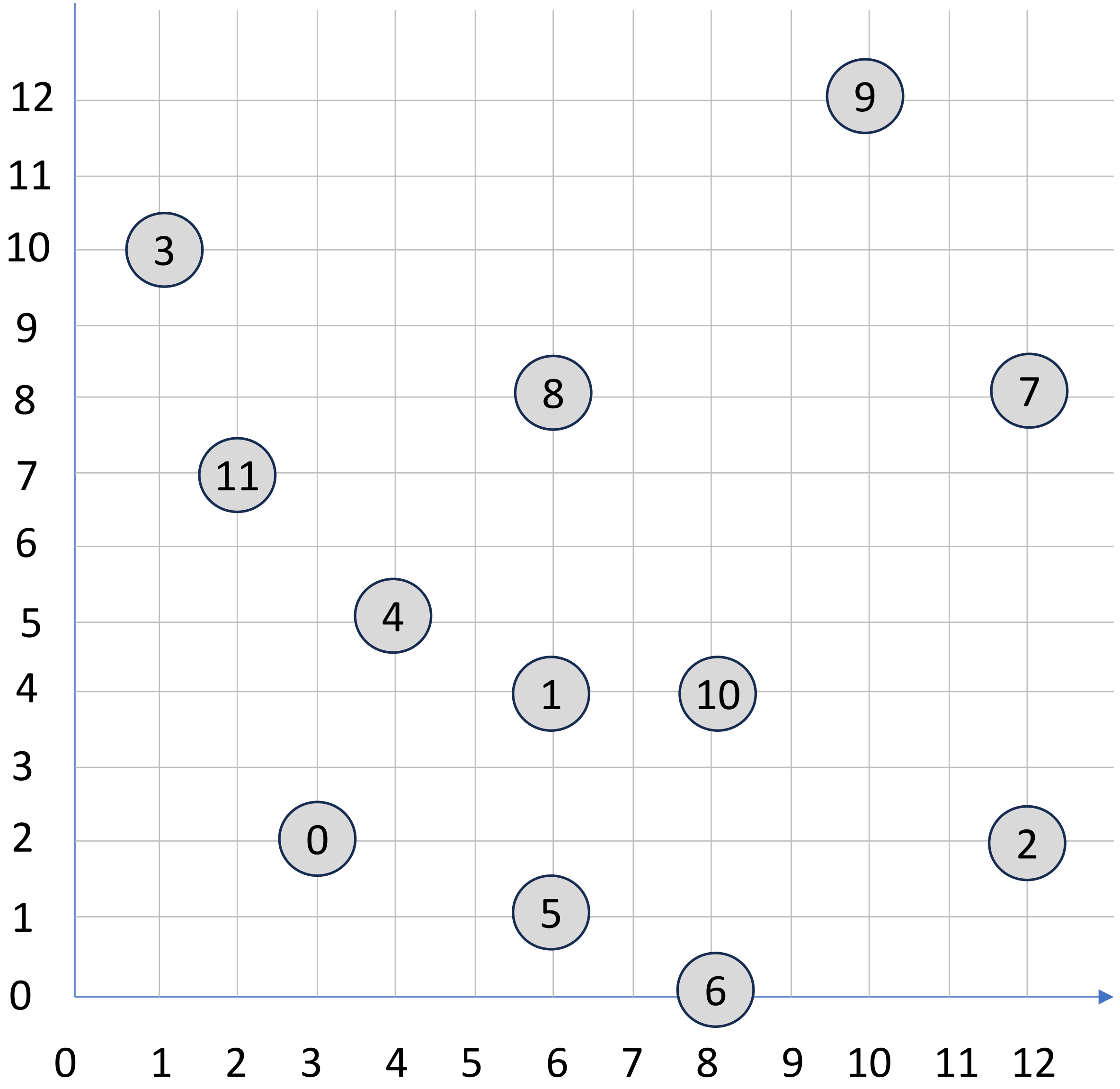
- Multiplication of 2 big numbers *A* and *B* (containing *n* digits)
- $A = A_1 x10^{n/2} + A_2$
- $B = B_1 x10^{n/2} + B_2$
- $A \text{ x } B = (A_1 x10^{n/2} + A_2) \text{ x } (B_1 x10^{n/2} + B_2) = A_1 \text{ x } B_1 x10^{n} + (A_1 \text{ x } B_2 + A_2 \text{ x } B_1) x10^{n/2} + A_2 \text{ x } B_2$
- $A_1 \text{ x } B_2 + A_2 \text{ x } B_1 = (A_1 + A_2) \text{ x } (B_1 + B_2) - A_1 \text{ x } B_1 - A_2 \text{ x } B_2$
- $A \text{ x } B = A_1 \text{ x } B_1 x10^{n} + ((A_1 + A_2) \text{ x } (B_1 + B_2) - A_1 \text{ x } B_1 - A_2 \text{ x } B_2) x10^{n/2} + A_2 \text{ x } B_2$

Multiplication of 2 big numbers: Karatsuba algorithm

- Multiplication of 2 big numbers *A* and *B* (containing *n* digits)
- $A = A_1 \text{x} 10^{n/2} + A_2$
- $B = B_1 \text{x} 10^{n/2} + B_2$
- $A \text{ x } B = (A_1 \text{x} 10^{n/2} + A_2) \text{ x } (B_1 \text{x} 10^{n/2} + B_2) = A_1 \text{ x } B_1 \text{x} 10^{n} + (A_1 \text{ x } B_2 + A_2 \text{ x } B_1) \text{x} 10^{n/2} + A_2 \text{ x } B_2$
- $A_1 \text{ x } B_2 + A_2 \text{ x } B_1 = (A_1 + A_2) \text{ x } (B_1 + B_2) - A_1 \text{ x } B_1 - A_2 \text{ x } B_2$
- $A \text{ x } B = A_1 \text{ x } B_1 \text{x} 10^{n} + ((A_1 + A_2) \text{ x } (B_1 + B_2) - A_1 \text{ x } B_1 - A_2 \text{ x } B_2) \text{x} 10^{n/2} + A_2 \text{ x } B_2$
- Complexity:
  - $T(n) = 3T(n/2) + O(n)$
  - $T(n) = O(n^{\log_2 3})$

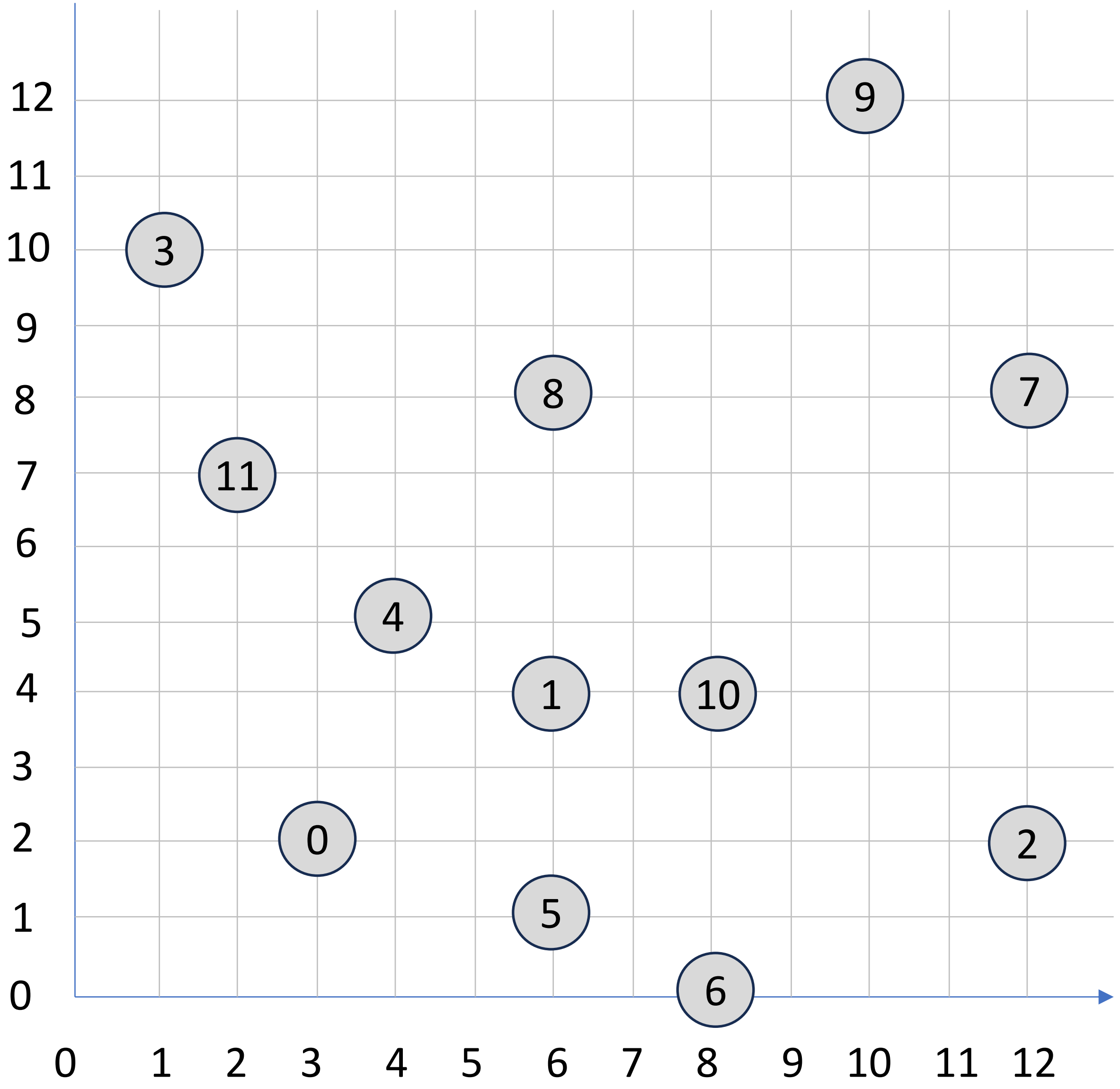
Closest pair of Points

- Given n points P = 0, 1, . . ., n-1 on the plane, find the pair of 2 points such that the distance between these points is the smallest.
- Denote A.x and A.y the x-coordinate and y-coordinate of point A.
- Denote dist(A, B): the distance between points A and B
- Denote d(P): the smallest distance among the distances between 2 points of P.



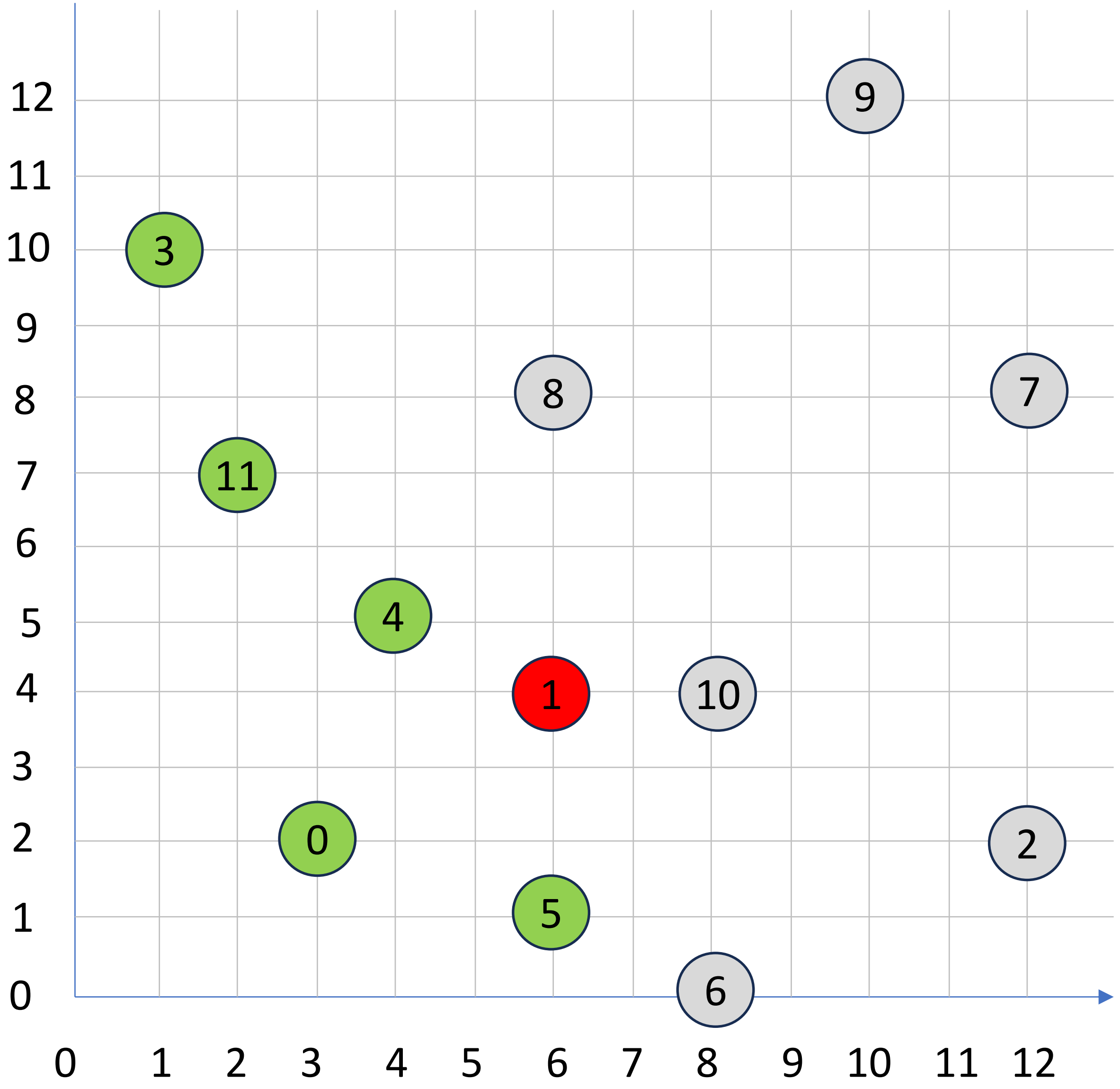
Closest pair of Points

- X_SORT(P): return the list of points of P sorted in a non-decreasing order of x-coordinates (two points with the same x-coordinate, the point having smaller y-coordinate will be located before the other)
- Y_SORT(P): return the list of points of P sorted in a non-decreasing order of y-coordinates (two points with the same y-coordinate, the point having smaller x-coordinate will be located before the other)
- Example
  - X_SORT(P) = 3, 11, 0, 4, 5, 1, 8, 6, 10, 9, 2, 7
  - Y_SORT(P) = 6, 5, 0, 2, 1, 10, 4, 11, 8, 7, 3, 9



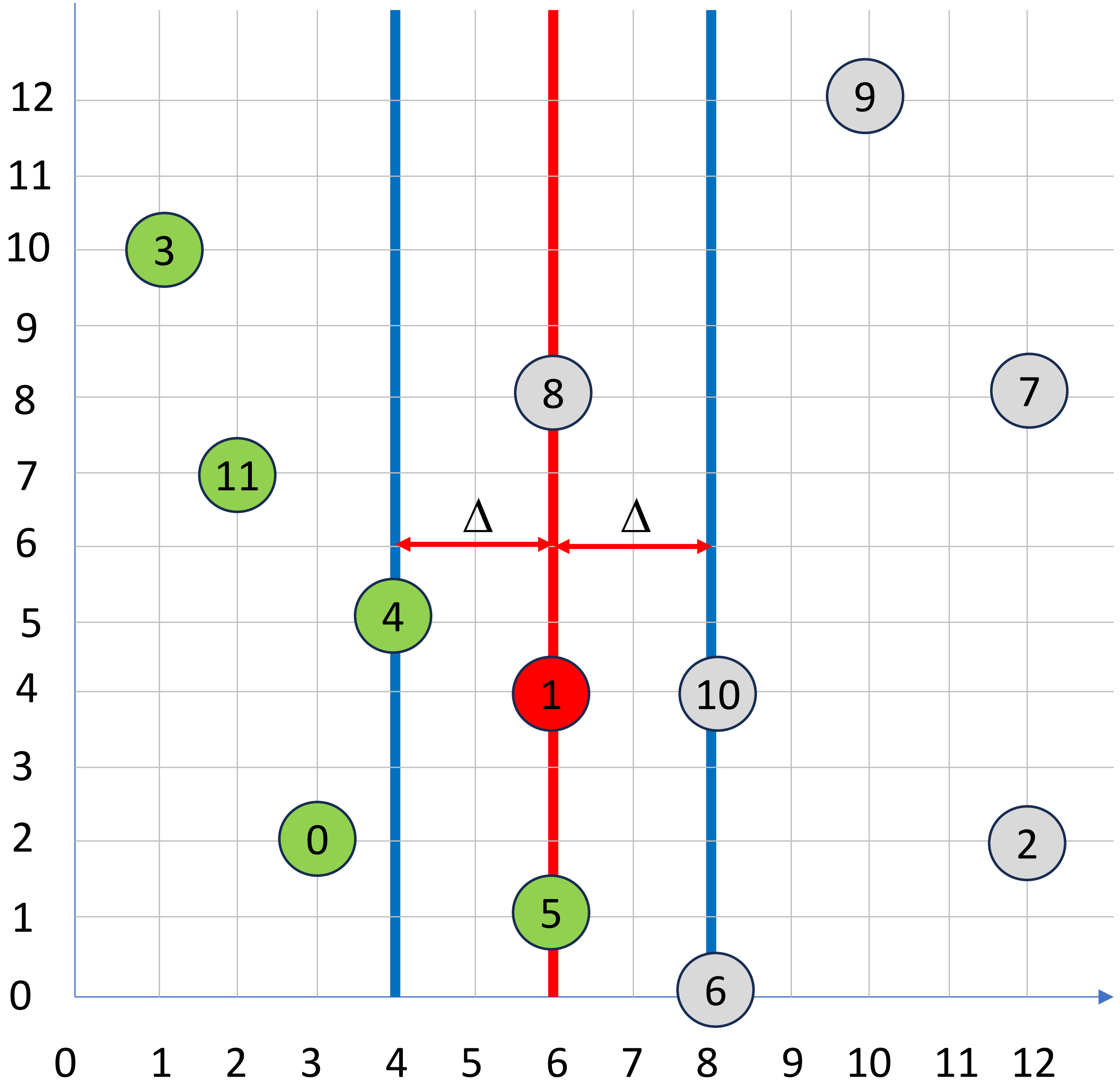
Closest pair of Points

- Let Px = X_SORT(P)
- Let O the point in the middle of Px.
- Let LEFT(Px, O) be the sub-list of points of Px before O (O inclusive)
- Let RIGHT(Px, O) be the sub-list of points of Px after O
- Example
  - Px = 3, 11, 0, 4, 5, 1, 8, 6, 10, 9, 2, 7
  - O = point 1
  - LEFT(Px, O) = 3, 11, 0, 4, 5, 1
  - RIGHT(Px, O) = 8, 6, 10, 9, 2, 7



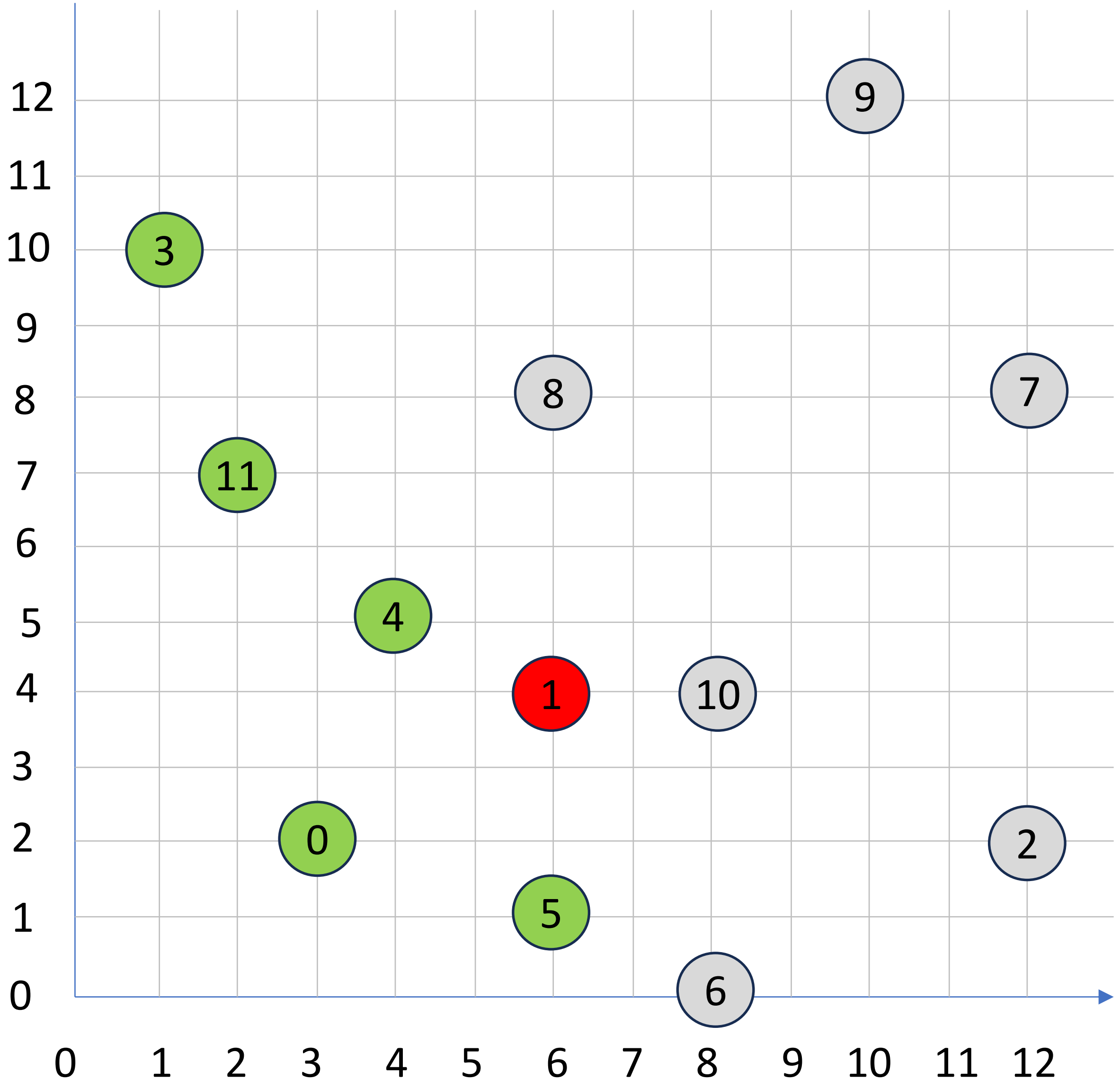
Closest pair of Points

- Divide and Conquer
  - Let Px = X_SORT(P)
  - Let O the point in the middle of Px.
  - PL = LEFT(Px, O)
  - PR = RIGHT(Px, O)
  - Let Δ = min(d(PL), d(PR))
  - Let S be the set of points A of Px such that |O.x – A.x| < Δ (S is called Strip).
  - Combination: find the closest points of S



Closest pair of Points

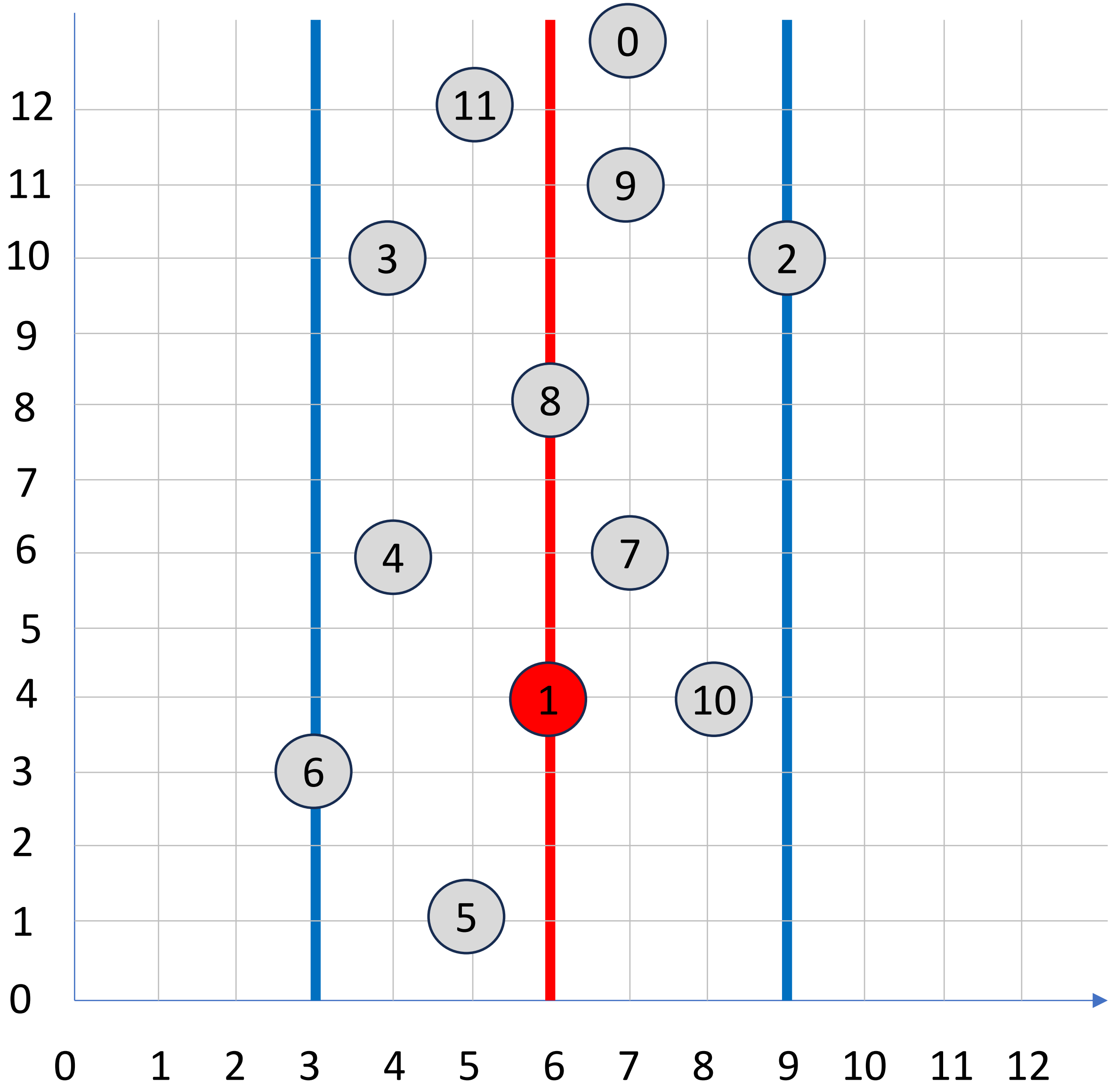
```
ClosestPair(P) {
  Px = X_SORT(P);
  n = length(P); O  = middle point of Px;
  PL = LEFT(Px, O); PR = RIGHT(Px, O);
  dL = ClosestPair(PL); dR = ClosestPair(PR);
  Δ = min(dL, dR);
  S = {A ∈ Px | Δ > |O.x – A.x|};
  dm = ClosestPairStrip(S, length(S), Δ);
  return dm;
}
```



Closest pair of Points

- Find the closest points in the strip

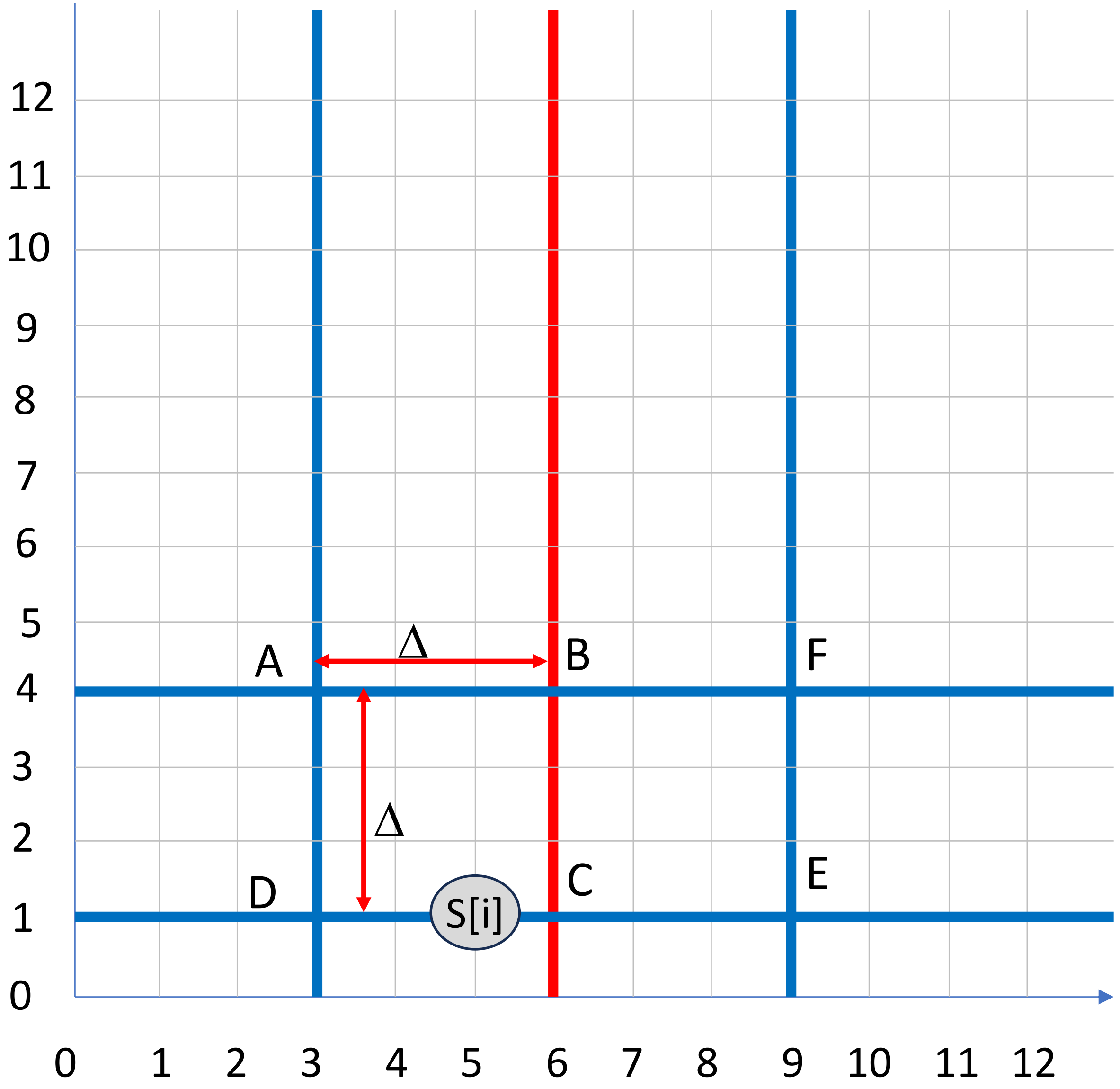
```
ClosestPairStrip(S, n, ∆) {
 S = Y_SORT(S);
 dm = ∆;
 for i = 0 to n-1 do {
  for j = i+1 to n-1 do {
    if S[j].y – S[i].y ≥ ∆ then break;
    dm = min(dm, dist(S[i], S[j]);
  }
 }
 return dm;
}
```



Closest pair of Points

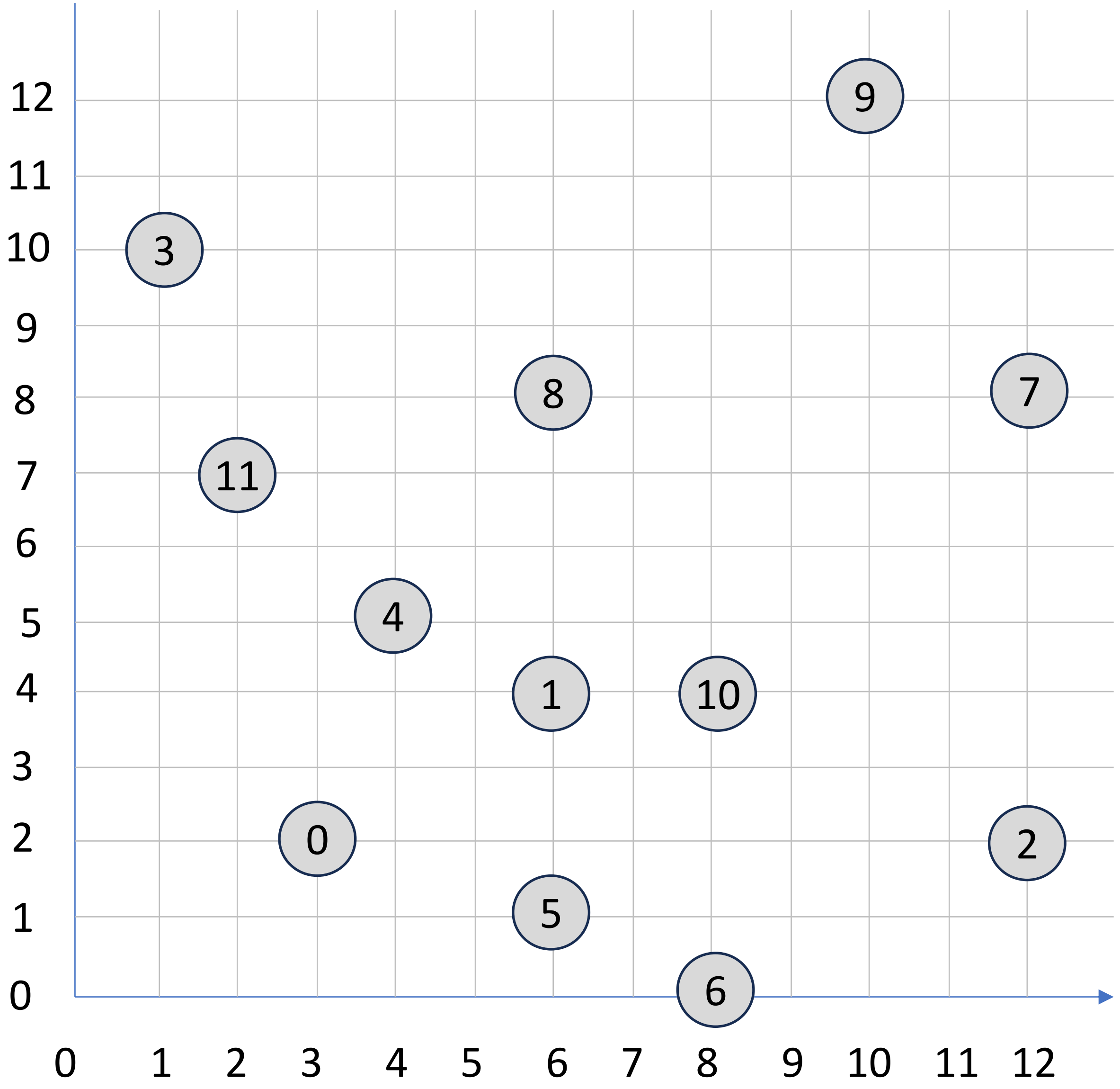
- Lines 5—8 run at most 8 iterations as each square ABCD and BCEF contains at most 4 points (distance between 2 points within each square is greater or equal to Δ)

```
1. ClosestPairStrip(S, n, Δ) {
2.  S = Y_SORT(S);
3.  dm = Δ;
4.  for i = 0 to n-1 do {
5.   for j = i+1 to n-1 do {
6.     if S[j].y – S[i].y ≥ Δ then break;
7.     dm = min(dm, dist(S[i], S[j]);
8.   }
9.  }
10. return dm;
11.}
```



Closest pair of Points – O(nlogn) implementation

- Px = X_SORT(P) and Py = Y_SORT(P)
- O is the middle point of Px
- PxL = LEFT(Px, O) and PxR = RIGHT(Px, O)
- PyL = Y_SORT(PxL) and PyR = Y_SORT(PxR)
- Example:
  - Px = 3, 11, 0, 4, 5, 1, 8, 6, 10, 9, 2, 7
  - Py = 6, 5, 0, 2, 1, 10, 4, 11, 8, 7, 3, 9
- Left part
  - PxL = 3, 11, 0, 4, 5, 1
  - PyL = 5, 0, 1, 4, 11, 3
- Right part
  - PxR = 8, 6, 10, 9, 2, 7
  - PyR = 6, 2, 10, 8, 7, 9



Closest pair of Points – O(nlogn) implementation

```
ClosestPair(P, n){// points of the list P are indexed 0, 1, . . ., n-1
  Px = X_SORT(P);
  Py = Y_SORT(P);
  dm = ClosestPair(Px, Py, n);
  return dm;
}
```

Closest pair of Points – O(nlogn) implementation

```
ClosestPair(Px, Py, n) {
  if n <= 3 then return BruteforceClosestPair(Px,n);
  PxL, PyL, PxR, PyR = [];  mid = n/2;  O = Px[mid];
  for i = 0 to mid – 1 do PxL.push(Px[i]);
  for i = mid to n-1 do PxR.push(Px[i]);
  for i = 0 to n-1 do {
   if ((Py[i].x < O.x) or Py[i].x = O.x and Py[i].y < O.y) and length(PyL) < mid then
      PyL.push(Py[i]);
    else PyR.push(Py[i]);
  }
  dL = ClosestPair(PxL, PyL, mid); dR = ClosestPair(PxR, PyR, n-mid);  Δ = min(dL, dR);
  S = []; for i = 0 to n-1 do if |Py[i].x – O.x| < Δ then S.push(Py[i]);
  dm = ClosestPairStrip(S, length(S), Δ);
  return dm;
}
```

Closest pair of Points – O(nlogn) implementation

```
ClosestPairStrip(S, n, Δ) {
  dm = Δ;
  for i = 0 to n-1 do {
    for j = i+1 to n-1 do {
      if S[j].y – S[i].y ≥ Δ then break;
      dm = min(dm, dist(S[i], S[j]);
    }
  }
  return dm;
}
```

```
BruteforceClosestPair(P, n){
  dm = INF;
  for i = 0 to n-2 do
    for j = i + 1 to n-1 do
      if dist(P[i], P[j]) < dm then
        dm = dist(P[i], P[j]);
  return dm;
}
```

Inversion

- Given a sequence $a[1], a[2], \ldots, a[n]$. Count the number of pairs (i, j) such that $1 \leq i < j \leq n$ and $a[i] > a[j]$
- Example: 5, 2, 7, 9, 4, 1
  - Inversions: (1, 2), (1, 5), (1, 6), (2, 6), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6)

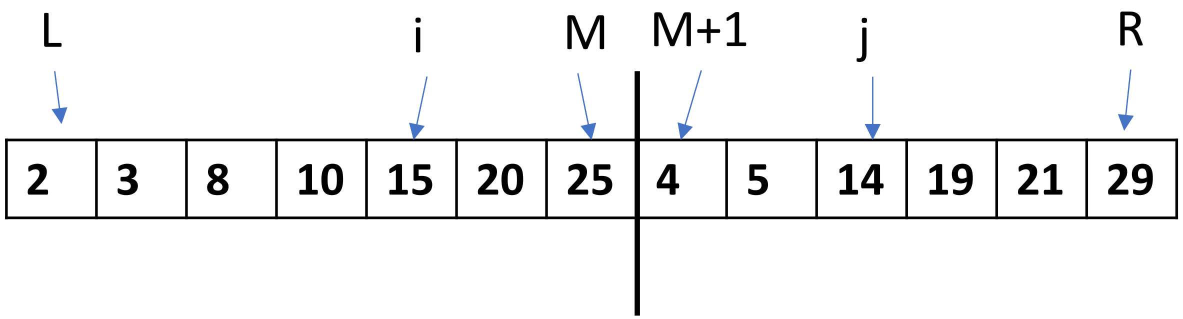
Inversion

- Divide and conquer: Apply merge sort algorithms
  - Divide the given sequence into 2 equal size parts
  - Count the number of inversions of the left sub-sequence (after counting, the left sub-sequence is sorted in a non-decreasing order)
  - Count the number of inversions of the right sub-sequence (after counting, the right sub-sequence is sorted in a non-decreasing order)
  - Count the number of pairs (*i*, *j*) in which *a*[*i*] > *a*[*j*] (*i* is an index of the left sub-sequence and *j* is an index of the right sub-sequence)

```
countInversions(L, R) {
  if L >= R then return 0;
  M = (L+R)/2;
  cntL = countInversions(L, M);
  cntR = countInversions(M+1, R);
  cnt = countMerge(L, M, R);
  return cntL + cntR + cnt;
}
```

Inversion

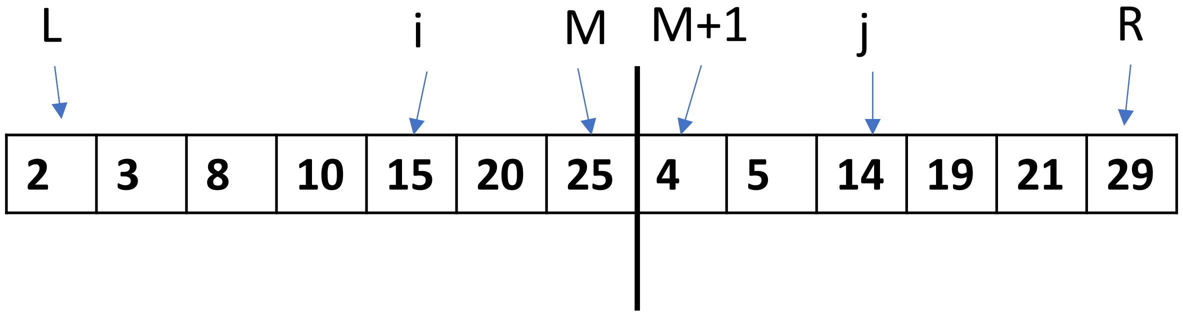
- Divide and conquer: Apply merge sort algorithms
- Merge operation:
  - Run an index *i* on the left sub-sequence and an index *j* on the right sub-sequence
  - If *a*[*i*] > *a*[*j*] then *a*[*q*] > *a*[*j*] for all *q* = *i* ,..., *M*: number of inversions is augmented by *M* – *i* + 1




```
countMerge(L, M, R) {
 i = L; j = M+1; cnt = 0;
 for k = L to R do {
  if(i > M) then {  ta[k] = a[j]; j++;  }
  else if(j > R) then { ta[k] = a[i]; i++; }
  else{
   if(a[i] <= a[j]) then {  ta[k] = a[i]; i++; }
   else{
    ta[k] = a[j]; j++;  cnt = cnt + M - i + 1;
   }
  }
 }
 for(int k = L; k <= R; k++) a[k] = ta[k];
 return cnt;
}
```

Inversion

- Divide and conquer: Apply merge sort algorithms
- Merge operation:
  - Run an index *i* on the left sub-sequence and an index *j* on the right sub-sequence
  - If $a[i] > a[j]$ then $a[q] > a[j]$ for all $q = i ,..., M$: number of inversions is augmented by $M - i + 1$
- Time complexity: O(*nlogn*)




```
countMerge(L, M, R) {
 i = L; j = M+1; cnt = 0;
 for k = L to R do {
  if(i > M) then {  ta[k] = a[j]; j++;  }
  else if(j > R) then { ta[k] = a[i]; i++; }
  else{
   if(a[i] <= a[j]) then {  ta[k] = a[i]; i++; }
   else{
    ta[k] = a[j]; j++;  cnt = cnt + M - i + 1;
   }
  }
 }
 for(int k = L; k <= R; k++) a[k] = ta[k];
 return cnt;
}
```

Decrease and Conquer

- Given a binary sequence *X* of length *n* which can be divided into 2 parts: the prefix contains only 0 and the suffix contains only 1.
  - Example: 0000000011111111111111111
- Goal: Find the index of the first 1-bit (from left to right)

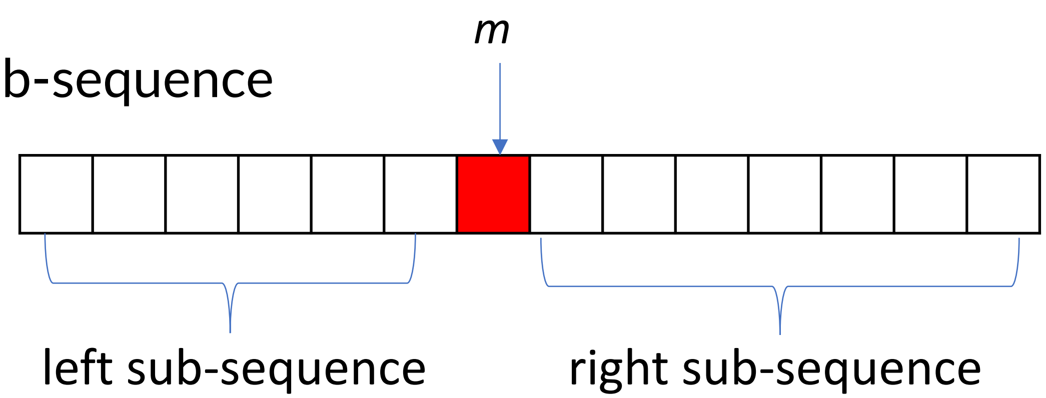
Decrease and Conquer

- Given a binary sequence *X* of length *n* which can be divided into 2 parts: the prefix contains only 0 and the suffix contains only 1.
  - Example: 000000001111111111111111
- Goal: Find the index of the first 1-bit (from left to right)
  - Example

0	0	0	0	0	0	0	0	0	1	1	1	1	1

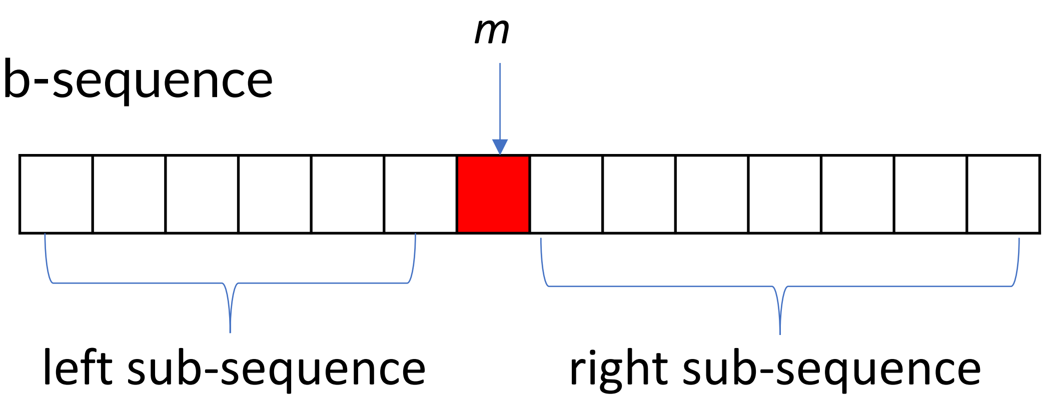
Decrease and Conquer

- Given a binary sequence X of length n which can be divided into 2 parts: the prefix contains only 0 and the suffix contains only 1.
  - Example: 0000000011111111111111111
- Goal: Find the index of the first 1-bit (from left to right)
- Decrease and conquer
  - Let m the middle position of X
  - Consider the bit $X[m]$ in the middle of X
    - If $X[m]$ = 0 then find in the result in the right sub-sequence
    - If $X[m]$ = 1
      - If $X[m$-1] = 0 then return m
      - Otherwise, find the result in the left sub-sequence



Decrease and Conquer

- Given a binary sequence *X* of length *n* which can be divided into 2 parts: the prefix contains only 0 and the suffix contains only 1.
  - Example: 000000001111111111111111
- Goal: Find the index of the first 1-bit (from left to right)
- Decrease and conquer
  - Let *m* the middle position of *X*
  - Consider the bit *X*[*m*] in the middle of *X*
    - If *X*[*m*] = 0 then find in the result in the right sub-sequence
    - If *X*[*m*] = 1
      - If *X*[*m*-1] = 0 then return *m*
      - Otherwise, find the result in the left sub-sequence
  - Time complexity: O(log*n*)



HUST

THANK YOU !

hust.edu.vn fb.com/dhbkhn