

## A. Pendahuluan

### 1. Paradigma Divide dan Conquer

Divide & Conquer merupakan teknik algoritmik dengan cara memecah input menjadi beberapa bagian, memecahkan masalah di setiap bagian secara rekursif, dan kemudian menggabungkan solusi untuk subproblem ini menjadi solusi keseluruhan. Menganalisis running time dari algoritma divide & conquer umumnya melibatkan penyelesaian rekurensi yang membatasi running time secara rekursif pada instance yang lebih kecil

### 2. Pengenalan Rekurensi

- Rekurensi adalah persamaan atau ketidaksetaraan yang menggambarkan fungsi terkait nilainya pada input yang lebih kecil. Ini adalah fungsi yang diekspresikan secara rekursif
- Ketika suatu algoritma berisi panggilan rekursif untuk dirinya sendiri, running time-nya sering dapat dijelaskan dengan perulangan
- Sebagai contoh, running time worst case  $T(n)$  dari algoritma merge-sort dapat dideskripsikan dengan perulangan:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

*with solution*  $T(n) = \Theta(n \log n)$

### 3. Bedah Algoritma Merge-Sort

- Merupakan algoritma sorting dengan paradigma divide & conquer
- Running time worst case-nya mempunyai laju pertumbuhan yang lebih rendah dibandingkan insertion sort
- Karena kita berhadapan dengan banyak subproblem, kita notasikan setiap subproblem sebagai sorting sebuah subarray  $A[p..r]$
- Inisialisasi,  $p=1$  dan  $r=n$ , tetapi nilai ini berubah selama kita melakukan perulangan subproblem

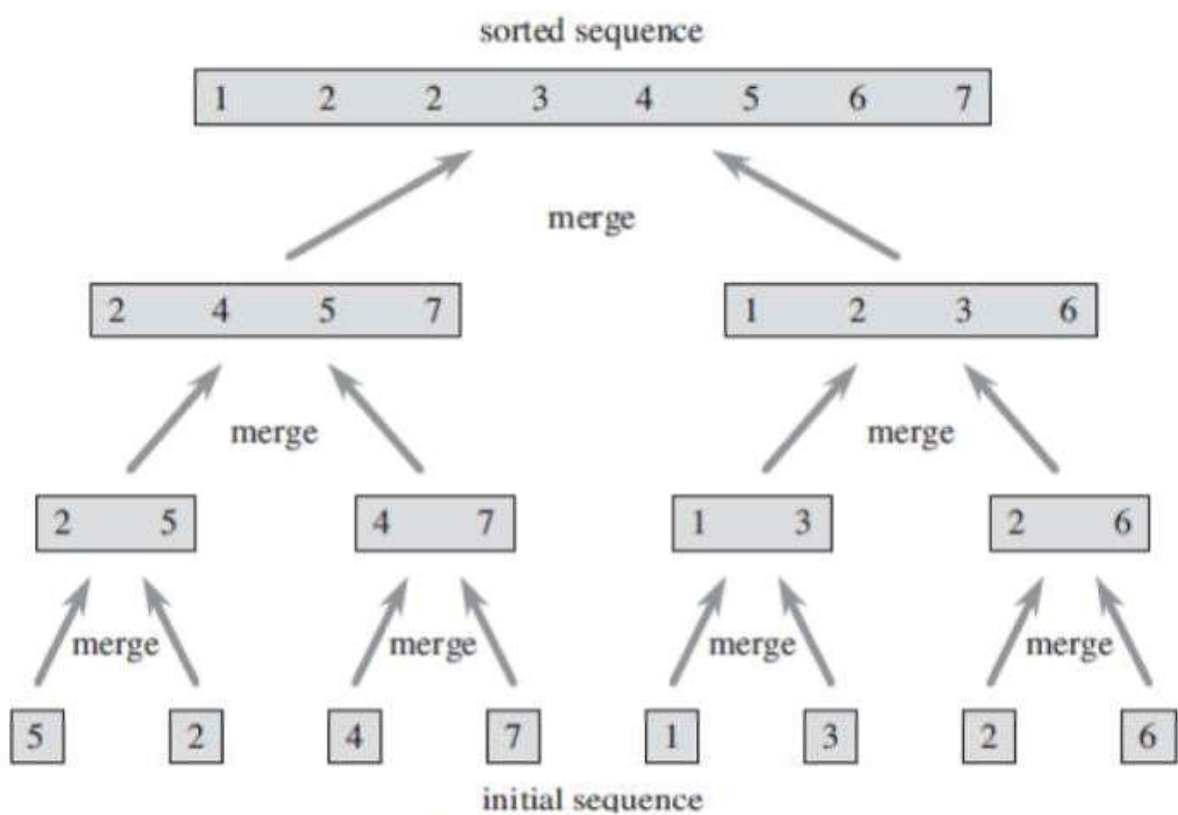
### 4. Untuk mengurutkan $A[p..r]$ :

- Divide dengan membagi input menjadi 2 subarray  $A[p..q]$  dan  $A[q+1 .. r]$
- Conquer dengan secara rekursif mengurutkan subarray  $A[p..q]$  dan  $A[q+1 .. r]$
- Combine dengan menggabungkan 2 subarray terurut  $A[p..q]$  dan  $A[q+1 .. r]$  untuk menghasilkan 1 subarray terurut  $A[p..r]$
- Untuk menyelesaikan langkah ini, kita membuat prosedur  $\text{MERGE}(A, p, q, r)$

- Rekursi berhenti apabila subarray hanya memiliki 1 elemen (secara trivial terurut)

#### 5. Pseudocode Merge-Sort

```
➤ MERGE-SORT(A, p, r)
  //sorts the elements in the subarray A[p..r]
  1  if p < r
  2    then q ← ⌊(p + r)/2⌋
  3      MERGE-SORT(A, p, q)
  4      MERGE-SORT(A, q + 1, r)
  5      MERGE(A, p, q, r)
```



Gambar 1. Ilustrasi algoritma merge-sort

#### 6. Prosedur Merge

- Prosedur merge berikut mengasumsikan bahwa subarray  $A[p..q]$  dan  $A[q+1 .. r]$  berada pada kondisi terurut. Prosedur merge menggabungkan kedua subarray untuk membentuk 1 subarray terurut yang menggantikan array saat ini  $A[p..r]$  (input).
- Ini membutuhkan waktu  $\Theta(n)$ , dimana  $n = r - p + 1$  adalah jumlah yang digabungkan
- Untuk menyederhanakan code, digunakanlah elemen sentinel (dengan nilai  $\infty$ ) untuk menghindari keharusan memeriksa apakah subarray kosong di setiap langkah dasar.

## 7. Pseudocode Prosedur Merge

```

MERGE(A, p, q, r)
1.  $n_1 \leftarrow q - p + 1$ ;  $n_2 \leftarrow r - q$ 
2. //create arrays L[1 ..  $n_1 + 1$ ] and R[1 ..  $n_2 + 1$ ]
3. for  $i \leftarrow 1$  to  $n_1$  do  $L[i] \leftarrow A[p + i - 1]$ 
4. for  $j \leftarrow 1$  to  $n_2$  do  $R[j] \leftarrow A[q + j]$ 
5.  $L[n_1 + 1] \leftarrow \infty$ ;  $R[n_2 + 1] \leftarrow \infty$ 
6.  $i \leftarrow 1$ ;  $j \leftarrow 1$ 
7. for  $k \leftarrow p$  to  $r$ 
8.   do if  $L[i] \leq R[j]$ 
9.     then  $A[k] \leftarrow L[i]$ 
10.         $i \leftarrow i + 1$ 
11.   else  $A[k] \leftarrow R[j]$ 
12.         $j \leftarrow j + 1$ 

```

## 8. Running Time Merge

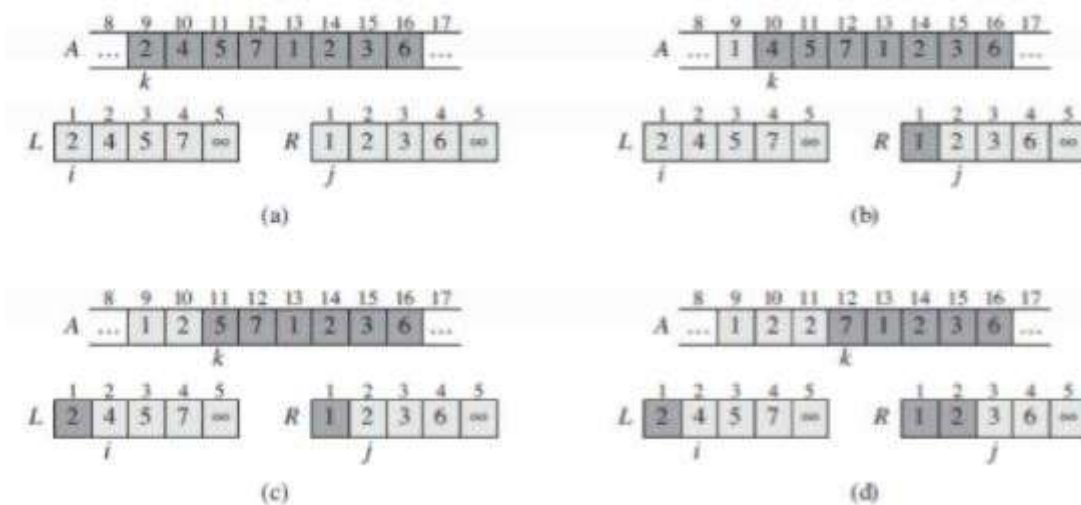
Untuk melihat running time prosedur MERGE berjalan di  $\Theta(n)$ , dimana  $n = p + 1$ , perhatikan perulangan for pada baris ke 3 dan 4,

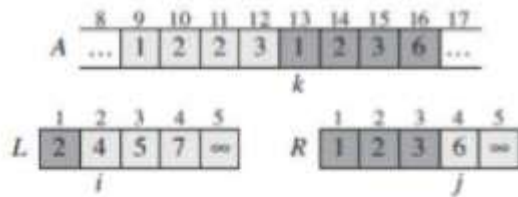
$$\Theta(n_1 + n_2) = \Theta(n)$$

dan ada sejumlah  $n$  iterasi pada baris ke 8-12 yang membutuhkan waktu konstan.

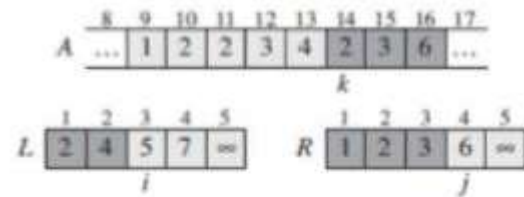
## 9. Contoh Sola Merge-Sort

MERGE(A, 9,12,16), dimana subarray  $A[9 \dots 16]$  mengandung sekuen (2,4,5,7,1,2,3,6)

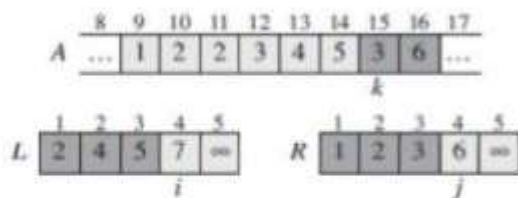




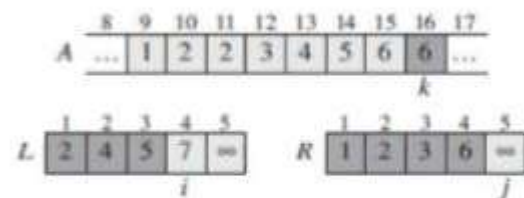
(e)



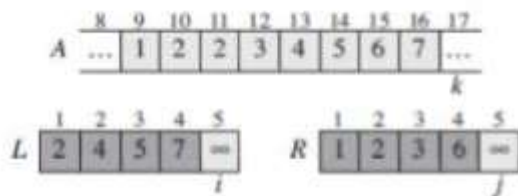
(f)



(g)



(h)



(i)

Algoritma merge-sort sangat mengikuti paradigma divide & conquer:

- Divide problem besar ke dalam beberapa subproblem
- Conquer subproblem dengan menyelesaikannya secara rekursif. Namun, apabila subproblem berukuran kecil, diselesaikan saja secara langsung
- Combine solusi untuk subproblem ke dalam solusi untuk original problem

Gunakan sebuah persamaan rekurensi (umumnya sebuah perulangan) untuk mendeskripsikan running time dari algoritma berparadigma divide & conquer.

$T(n)$  = running time dari sebuah algoritma berukuran  $n$

- Jika ukuran problem cukup kecil (misalkan  $n \leq c$ , untuk nilai  $c$  konstan), kita mempunyai best case. Solusi brute-force membutuhkan waktu konstan  $\Theta(1)$
- Sebailknya, kita membagi input ke dalam sejumlah  $a$  subproblem, setiap  $(1/b)$  dari ukuran original problem (Pada merge sort  $a = b = 2$ )
- Misalkan waktu yang dibutuhkan untuk membagi ke dalam  $n$ -ukuran problem adalah  $D(n)$

- Ada sebanyak  $a$  subproblem yang harus diselesaikan, setiap subproblem  $(n/b) \rightarrow$  setiap subproblem membutuhkan waktu  $T(n/b)$  sehingga kita menghabiskan  $aT(n/b)$
- Waktu untuk combine solusi kita misalkan  $C(n)$
- Maka persamaan rekurensinya untuk divide & conquer adalah:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Setelah mendapatkan rekurensi dari sebuah algoritma divide & conquer, selanjutnya rekurensi harus diselesaikan untuk dapat menentukan kompleksitas waktu asimptotiknya. Penyelesaian rekurensi dapat menggunakan 3 cara yaitu, metode substitusi, metode recursion-tree dan metode master. Ketiga metode ini dapat dilihat pada slide yang diberikan.

## B. Studi Kasus

### 1. Studi Kasus 5: Mencari Pasangan Titik Terdekat (Closest Pair of Points)

Identifikasi Problem:

Diberikan array  $n$  poin dalam bidang kartesius, dan problemnya adalah mencari tahu pasangan poin terdekat dalam bidang tersebut dengan merepresentasikannya ke dalam array. Masalah ini muncul di sejumlah aplikasi. Misalnya, dalam kontrol lalu lintas udara, kita mungkin ingin memantau pesawat yang terlalu berdekatan karena ini mungkin menunjukkan kemungkinan tabrakan. Ingat rumus berikut untuk jarak antara dua titik  $p$  dan  $q$ .

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Solusi:

Solusi umum dari permasalahan tersebut adalah menggunakan algoritma Brute force dengan  $O(n^2)$ , hitung jarak antara setiap pasangan dan kembalikan yang terkecil. Namun, kita dapat menghitung jarak terkecil dalam waktu  $O(n \log n)$  menggunakan strategi Divide and Conquer.

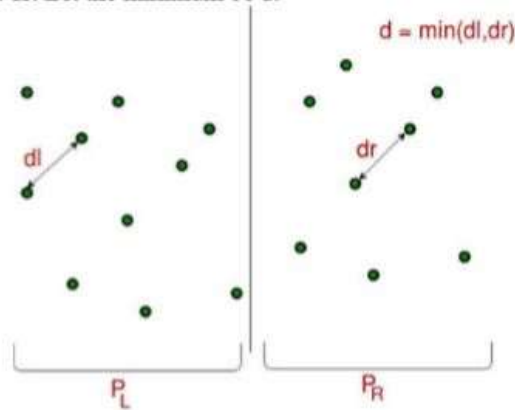
Ikuti algoritma berikut:

*Input:* An array of  $n$  points  $P[]$

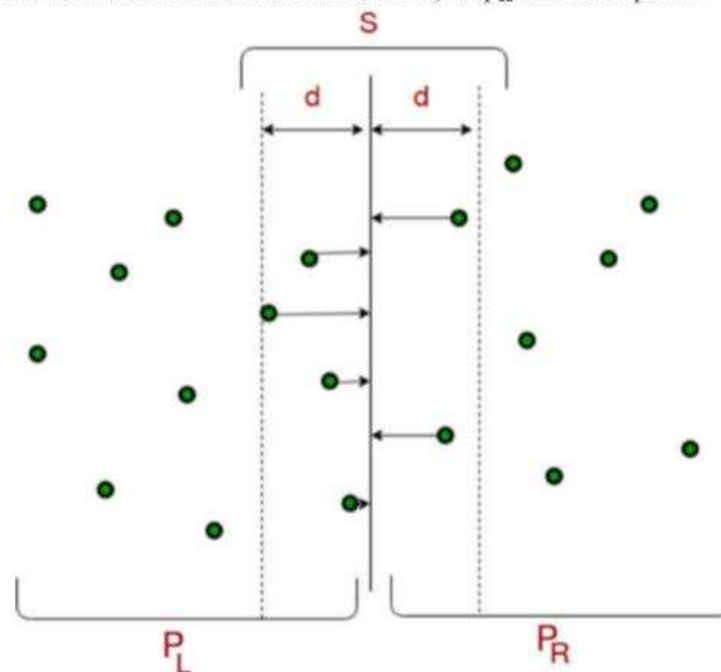
*Output:* The smallest distance between two points in the given array.

As a pre-processing step, input array is sorted according to  $x$  coordinates.

- 1) Find the middle point in the sorted array, we can take  $P[n/2]$  as middle point.
- 2) Divide the given array in two halves. The first subarray contains points from  $P[0]$  to  $P[n/2]$ . The second subarray contains points from  $P[n/2+1]$  to  $P[n-1]$ .
- 3) Recursively find the smallest distances in both subarrays. Let the distances be  $d_l$  and  $d_r$ . Find the minimum of  $d_l$  and  $d_r$ . Let the minimum be  $d$ .



- 4) From above 3 steps, we have an upper bound  $d$  of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through  $P[n/2]$  and find all points whose  $x$  coordinate is closer than  $d$  to the middle vertical line. Build an array  $strip[]$  of all such points.



- 5) Sort the array  $strip[]$  according to  $y$  coordinates. This step is  $O(n \log n)$ . It can be optimized to  $O(n)$  by recursively sorting and merging.
- 6) Find the smallest distance in  $strip[]$ . This is tricky. From first look, it seems to be a  $O(n^2)$  step, but it is actually  $O(n)$ . It can be proved geometrically that for every point in  $strip$ , we only need to check at most 7 points after it (note that  $strip$  is sorted according to  $Y$  coordinate). See [this](#) for more analysis.
- 7) Finally return the minimum of  $d$  and distance calculated in above step (step 6)

Fadlan Mulya Priatna

140810180041

Tugas 5

Tugas:

- Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

```
/*
Nama Program          : Closest Pair of Points
Nama                  : Fadlan Mulya Priatna
NPM                   : 140810180041
Tanggal Pembuatan     : 30 Maret 2020
Deskripsi Program    : Program untuk menyelesaikan problem closest pair of points
menggunakan algoritma divide & conquer yang diberikan dengan bahasa C++
*/

#include <iostream>
#include <cmath>

using namespace std;

struct point{
    double x, y;
};

int compareX(const void* a, const void* b){
    point *p1 = (point *)a, *p2 = (point *)b;
    return (p1->x - p2->x);
}

int compareY(const void* a, const void* b){
    point *p1 = (point *)a, *p2 = (point *)b;
    return (p1->y - p2->y);
}

float dist(point p1, point p2){
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

float small_dist(point P[], int n){
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i){
        for (int j = i + 1; j < n; ++j){
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
        }
    }
    return min;
}

float stripClosest(point strip[], int size, float d){
```

```
float min = d;
for (int i = 0; i < size; ++i){
    for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j){
        if (dist(strip[i],strip[j]) < min)
            min = dist(strip[i], strip[j]);
    }
}
return min;
}
```

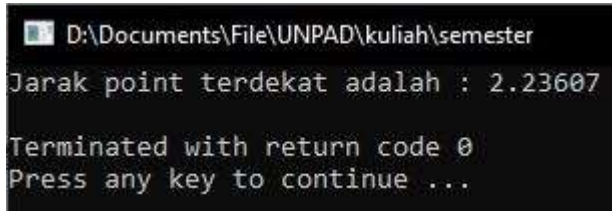
```
float closestUtil(point Px[], point Py[], int n){
    if (n <= 3)
        return small_dist(Px, n);
    int mid = n / 2;
    point midPoint = Px[mid];
    point Pyl[mid + 1];
    point Pyr[n - mid - 1];
    int li = 0, ri = 0;
    for (int i = 0; i < n; i++){
        if (Py[i].x <= midPoint.x)
            Pyl[li++] = Py[i];
        else
            Pyr[ri++] = Py[i];
    }
    float dl = closestUtil(Px, Pyl, mid);
    float dr = closestUtil(Px + mid, Pyr, n-mid);
    float d = min(dl, dr);
    point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++){
        if (abs(Py[i].x - midPoint.x) < d)
            strip[j] = Py[i], j++;
    }
    return min(d, stripClosest(strip, j, d));
}
```

```
float closest(point P[], int n){
    point Px[n];
    point Py[n];
    for (int i = 0; i < n; i++){
        Px[i] = P[i];
        Py[i] = P[i];
    }
    qsort(Px, n, sizeof(point), compareX);
    qsort(Py, n, sizeof(point), compareY);
    return closestUtil(Px, Py, n);
}
```

```
int main(){
```



```
point P[] = {{2, 5}, {15, 30}, {42, 50}, {5, 2}, {12, 13}, {3, 7}};  
int n = sizeof(P) / sizeof(P[0]);  
cout << "Jarak point terdekat adalah : " << closest(P, n);  
return 0;  
}
```



```
D:\Documents\File\UNPAD\kuliah\semester  
Jarak point terdekat adalah : 2.23607  
Terminated with return code 0  
Press any key to continue ...
```

- Tentukan rekurensi dari algoritma tersebut, dan selesaikan rekurensinya menggunakan metode recursion tree untuk membuktikan bahwa algoritma tersebut memiliki Big-O ( $n \log n$ )

Jawab:

Kompleksitas Waktu

Biarkan kompleksitas waktu dari algoritma di atas menjadi  $T(n)$ . Mari kita asumsikan bahwa kita menggunakan algoritma pengurutan  $O(n \log n)$ . Algoritma di atas membagi semua titik dalam dua set dan secara rekursif memanggil dua set. Setelah membelah, ia menemukan strip dalam waktu  $O(n)$ , mengurutkan strip dalam waktu  $O(n \log n)$  dan akhirnya menemukan titik terdekat dalam strip dalam waktu  $O(n)$ . Jadi  $T(n)$  dapat dinyatakan sebagai berikut

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) + O(n \log n) + O(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$

$$T(n) = T(n \times \log n \times \log n)$$

Catatan:

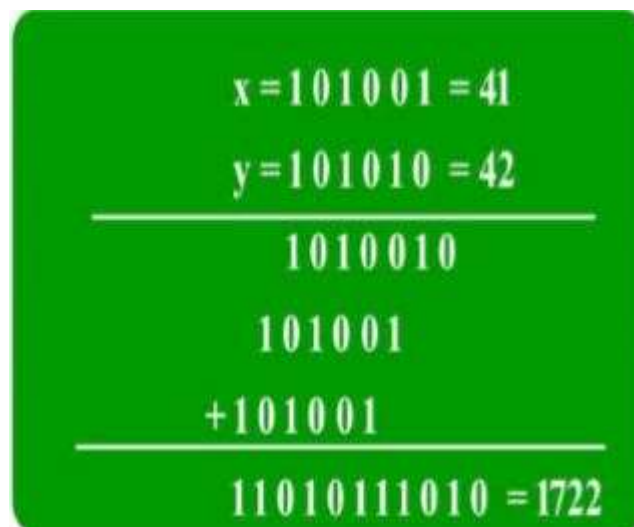
- a. Kompleksitas waktu dapat ditingkatkan menjadi  $O(n \log n)$  dengan mengoptimalkan langkah 5 dari algoritma di atas.
- b. Kode menemukan jarak terkecil. Dapat dengan mudah dimodifikasi untuk menemukan titik dengan jarak terkecil.
- c. Kode ini menggunakan pengurutan cepat yang bisa  $O(n^2)$  dalam kasus terburuk. Untuk memiliki batas atas sebagai  $O(n(\log n)^2)$ , algoritma pengurutan  $O(n \log n)$  seperti pengurutan gabungan atau pengurutan tumpukan dapat digunakan

## 2. Studi Kasus 6: Algoritma Karatsuba untuk Perkalian Cepat

Identifikasi Problem: Diberikan dua string biner yang mewakili nilai dua bilangan bulat, cari produk (hasil kali) dari dua string. Misalnya, jika string bit pertama adalah "1100" dan string bit kedua adalah "1010", output harus 120. Supaya lebih sederhana, panjang dua string sama dan menjadi  $n$ .

Solusi:

Salah satu solusinya adalah dengan naïve approach yang pernah kita pelajari di sekolah. Satu per satu ambil semua bit nomor kedua dan kalikan dengan semua bit nomor pertama. Akhirnya tambahkan semua perkalian. Algoritma ini membutuhkan waktu  $O(n^2)$ .


$$\begin{array}{r} x = 101001 = 41 \\ y = 101010 = 42 \\ \hline 1010010 \\ 101001 \\ + 101001 \\ \hline 11010111010 = 1722 \end{array}$$

Algoritma Karatsuba

Solusi lain adalah dengan menggunakan Algoritma Karatsuba yang berparadigma Divide dan Conquer, kita dapat melipatgandakan dua bilangan bulat dalam kompleksitas waktu yang lebih sedikit. Kami membagi angka yang diberikan dalam dua bagian. Biarkan angka yang diberikan menjadi  $X$  dan  $Y$ .

- Untuk kesederhanaan, kita asumsikan bahwa  $n$  adalah genap:

$X = X_l \cdot 2^{n/2} + X_r$	[ $X_l$ and $X_r$ contain leftmost and rightmost $n/2$ bits of $X$ ]
$Y = Y_l \cdot 2^{n/2} + Y_r$	[ $Y_l$ and $Y_r$ contain leftmost and rightmost $n/2$ bits of $Y$ ]

- Produk  $XY$  dapat ditulis sebagai berikut:

$\begin{aligned} XY &= (X_l \cdot 2^{n/2} + X_r) (Y_l \cdot 2^{n/2} + Y_r) \\ &= 2^n X_l Y_l + 2^{n/2} (X_l Y_r + X_r Y_l) + X_r Y_r \end{aligned}$
---

- Jika kita melihat rumus di atas, ada empat perkalian ukuran  $n/2$ , jadi pada dasarnya kita membagi masalah ukuran  $n$  menjadi empat sub-masalah ukuran  $n/2$ . Tetapi itu tidak membantu karena solusi pengulangan  $T(n) = 4T(n/2) + O(n)$  adalah  $O(n^2)$ . Bagian rumit dari algoritma ini adalah mengubah dua istilah tengah ke bentuk lain sehingga hanya satu perkalian tambahan yang cukup. Berikut ini adalah tricky expression untuk dua middle terms tersebut.

$$X_l Y_r + X_r Y_l = (X_l + X_r)(Y_l + Y_r) - X_l Y_l - X_r Y_r$$

- Jadi nilai akhir XY menjadi:

$$XY = 2^n X_l Y_l + 2^{n/2} * [(X_l + X_r)(Y_l + Y_r) - X_l Y_l - X_r Y_r] + X_r Y_r$$

- Dengan trik di atas, perulangan menjadi  $T(n) = 3T(n/2) + O(n)$  dan solusi dari perulangan ini adalah  $O(n^{1.59})$

$$XY = 2^{2\lceil n/2 \rceil} X_l Y_l + 2^{\lceil n/2 \rceil} * [(X_l + X_r)(Y_l + Y_r) - X_l Y_l - X_r Y_r] + X_r Y_r$$

Bagaimana jika panjang string input berbeda dan tidak genap? Untuk menangani kasus panjang yang berbeda, kita menambahkan 0 di awal. Untuk menangani panjang ganjil, kita menempatkan bit floor ( $n/2$ ) di setengah kiri dan ceil ( $n/2$ ) bit di setengah kanan. Jadi ekspresi untuk XY berubah menjadi berikut.

$$XY = 2^{2\lceil n/2 \rceil} X_l Y_l + 2^{\lceil n/2 \rceil} * [(X_l + X_r)(Y_l + Y_r) - X_l Y_l - X_r Y_r] + X_r Y_r$$

Tugas:

- Buatlah program untuk menyelesaikan problem fast multiplication menggunakan algoritma divide & conquer yang diberikan (Algoritma Karatsuba). Gunakan bahasa C++

/\*

Nama Program : Algoritma Karatsuba

Nama : Fadlan Mulya Priatna

NPM : 140810180041

Tanggal Pembuatan : 30 Maret 2020

Deskripsi Program : program untuk menyelesaikan problem fast multiplication menggunakan algoritma divide & conquer yang diberikan (Algoritma Karatsuba). Dalam bahasa C++

```
*/

#include<iostream>
#include<stdio.h>

using namespace std;

// FOLLOWING TWO FUNCTIONS ARE COPIED FROM http://goo.gl/q0OhZ
// Helper method: given two unequal sized bit strings, converts them to
// same length by adding leading 0s in the smaller string. Returns the
// the new length

int makeEqualLength(string &str1, string &str2){
    int len1 = str1.size();
    int len2 = str2.size();
    if (len1 < len2){
        for (int i = 0 ; i < len2 - len1 ; i++)
            str1 = '0' + str1;
        return len2;
    }
    else if (len1 > len2){
        for (int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2;
    }
    return len1; // If len1 >= len2
}

// The main function that adds two bit sequences and returns the addition
string addBitStrings( string first, string second ){
    string result; // To store the sum bits

    // make the lengths same before adding
    int length = makeEqualLength(first, second);
    int carry = 0; // Initialize carry

    // Add all bits one by one
    for (int i = length-1 ; i >= 0 ; i--){
        int firstBit = first.at(i) - '0';
        int secondBit = second.at(i) - '0';

        // boolean expression for sum of 3 bits
        int sum = (firstBit ^ secondBit ^ carry)+'0';

        result = (char)sum + result;

        // boolean expression for 3-bit addition
        carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
    }
}
```

Fadlan Mulya Priatna  
140810180041  
Tugas 5

```
// if overflow, then add a leading 1
if (carry) result = '1' + result;

return result;
}

// A utility function to multiply single bits of strings a and b
int multiplyiSingleBit(string a, string b){
    return (a[0] - '0')*(b[0] - '0');
}

// The main function that multiplies two bit strings X and Y and returns
// result as long integer
long int multiply(string X, string Y){
    // Find the maximum of lengths of x and Y and make length
    // of smaller string same as that of larger string
    int n = makeEqualLength(X, Y);

    // Base cases
    if (n == 0) return 0;
    if (n == 1) return multiplyiSingleBit(X, Y);

    int fh = n/2; // First half of string, floor(n/2)
    int sh = (n-fh); // Second half of string, ceil(n/2)

    // Find the first half and second half of first string.
    // Refer http://goo.gl/ILmgn for substr method
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);

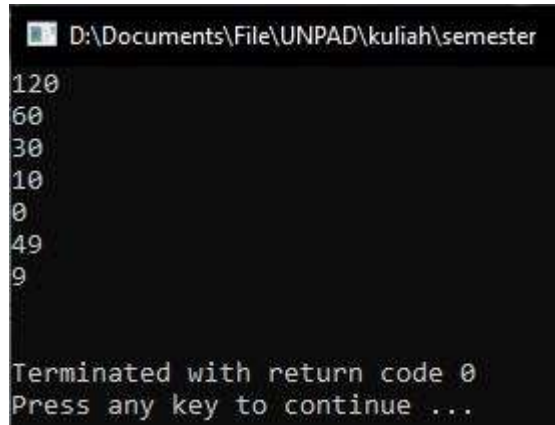
    // Find the first half and second half of second string
    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);

    // Recursively calculate the three products of inputs of size n/2
    long int P1 = multiply(Xl, Yl);
    long int P2 = multiply(Xr, Yr);
    long int P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));

    // Combine the three products to get the final result.
    return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}

// Driver program to test aboev functions
int main(){
    printf ("%ld\n", multiply("1100", "1010"));
    printf ("%ld\n", multiply("110", "1010"));
    printf ("%ld\n", multiply("11", "1010"));
    printf ("%ld\n", multiply("1", "1010"));
}
```

```
printf ("%ld\n", multiply("0", "1010"));
printf ("%ld\n", multiply("111", "111"));
printf ("%ld\n", multiply("11", "11"));
}
```



```
D:\Documents\File\UNPAD\kuliah\semester
120
60
30
10
0
49
9
Terminated with return code 0
Press any key to continue ...
```

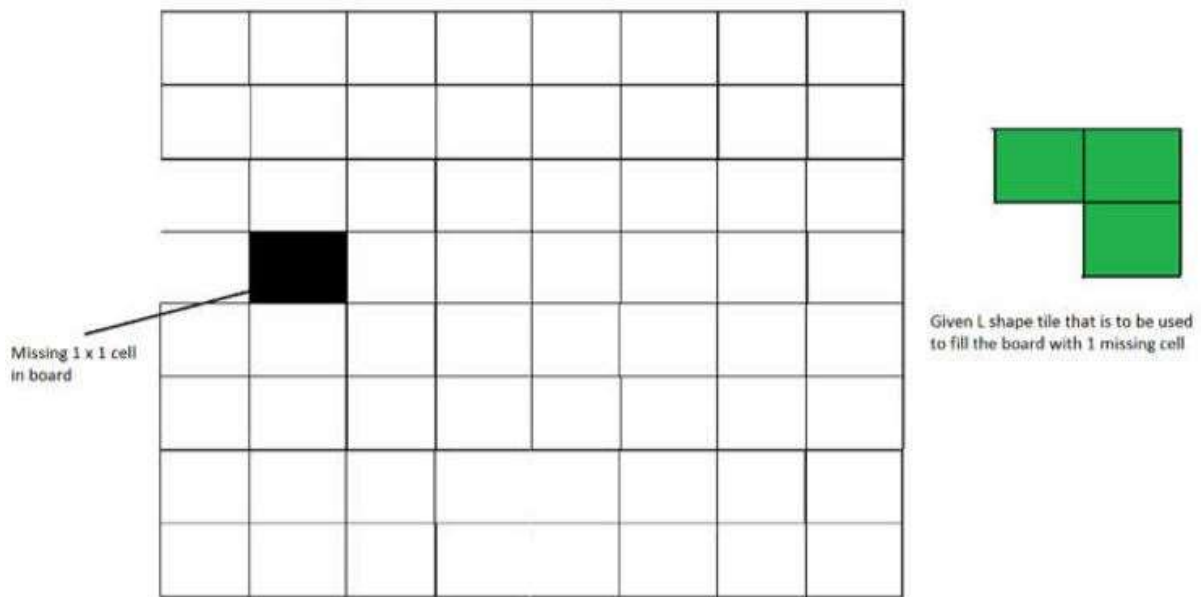
- Rekurensi dari algoritma tersebut adalah  $T(n) = 3T(n/2) + O(n)$ , dan selesaikan rekurensinya menggunakan metode substitusi untuk membuktikan bahwa algoritma tersebut memiliki Big-O ( $n \log n$ )

- Let's try divide and conquer.
  - Divide each number into two halves.
    - $x = x_H r^{n/2} + x_L$
    - $y = y_H r^{n/2} + y_L$
  - Then:
    - $xy = (x_H r^{n/2} + x_L) (y_H r^{n/2} + y_L)$
    - $= x_H y_H r^n + (x_H y_L + x_L y_H) r^{n/2} + x_L y_L$
  - Runtime?
    - $T(n) = 4 T(n/2) + O(n)$
    - $T(n) = O(n^2)$
- Instead of 4 subproblems, we only need 3 (with the help of clever insight).
- Three subproblems:
  - $a = x_H y_H$
  - $d = x_L y_L$
  - $e = (x_H + x_L) (y_H + y_L) - a - d$
- Then  $xy = a r^n + e r^{n/2} + d$
- $T(n) = 3 T(n/2) + O(n)$
- $T(n) = O(n^{\log 3}) = O(n^{1.584...})$

### 3. Studi Kasus 7: Permasalahan Tata Letak Keramik Lantai (Tiling Problem)

Identifikasi Problem: Diberikan papan berukuran  $n \times n$  dimana  $n$  adalah dari bentuk  $2k$  dimana  $k \geq 1$  (Pada dasarnya  $n$  adalah pangkat dari 2 dengan nilai minimumnya 2). Papan memiliki satu sel yang hilang (ukuran  $1 \times 1$ ). Isi papan menggunakan ubin

berbentuk L. Ubin berbentuk L berukuran  $2 \times 2$  persegi dengan satu sel berukuran  $1 \times 1$  hilang.



Gambar 2. Ilustrasi tiling problem

Solusi:

Masalah ini dapat diselesaikan menggunakan Divide and Conquer. Di bawah ini adalah algoritma rekursifnya

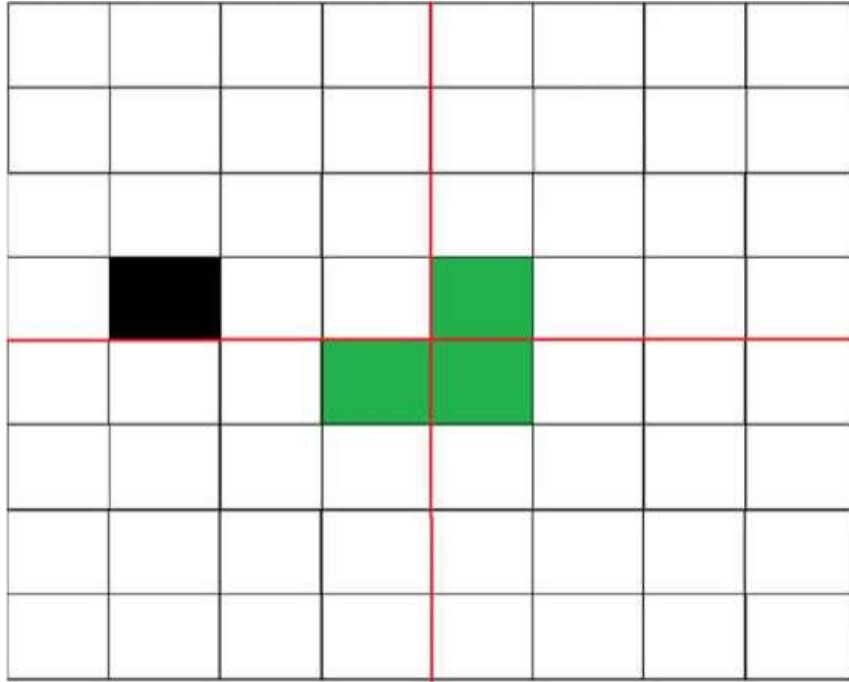
```
// n is size of given square, p is location of missing cell
Tile(int n, Point p)

1) Base case: n = 2, A 2 x 2 square with one cell missing is nothing
   but a tile and can be filled with a single tile.

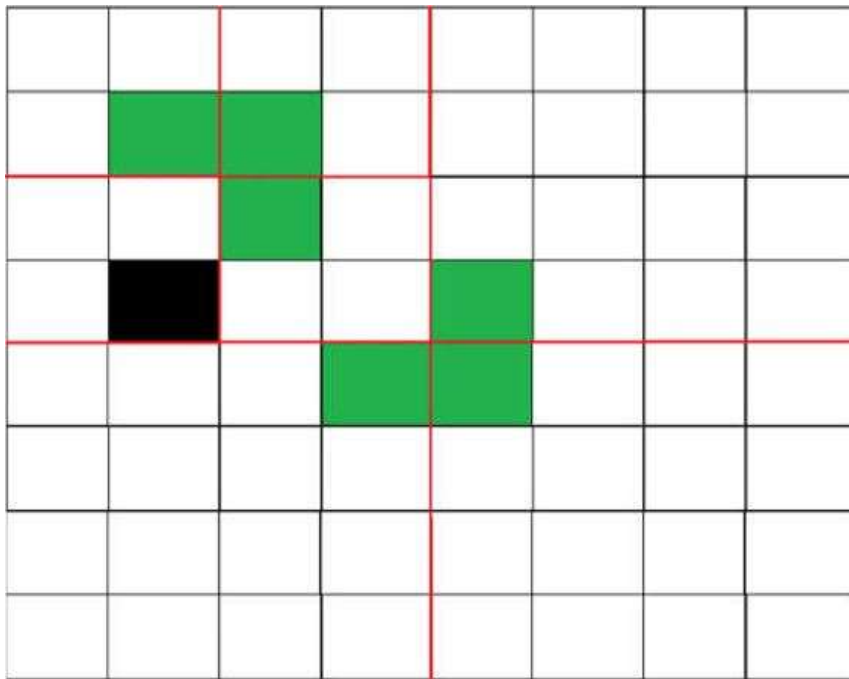
2) Place a L shaped tile at the center such that it does not cover
   the n/2 * n/2 subsquare that has a missing square. Now all four
   subsquares of size n/2 x n/2 have a missing cell (a cell that doesn't
   need to be filled). See figure 3 below.

3) Solve the problem recursively for following four. Let p1, p2, p3 and
   p4 be positions of the 4 missing cells in 4 squares.
   a) Tile(n/2, p1)
   b) Tile(n/2, p2)
   c) Tile(n/2, p3)
   d) Tile(n/2, p3)
```

Gambar di bawah ini menunjukkan kerja algoritma di atas.

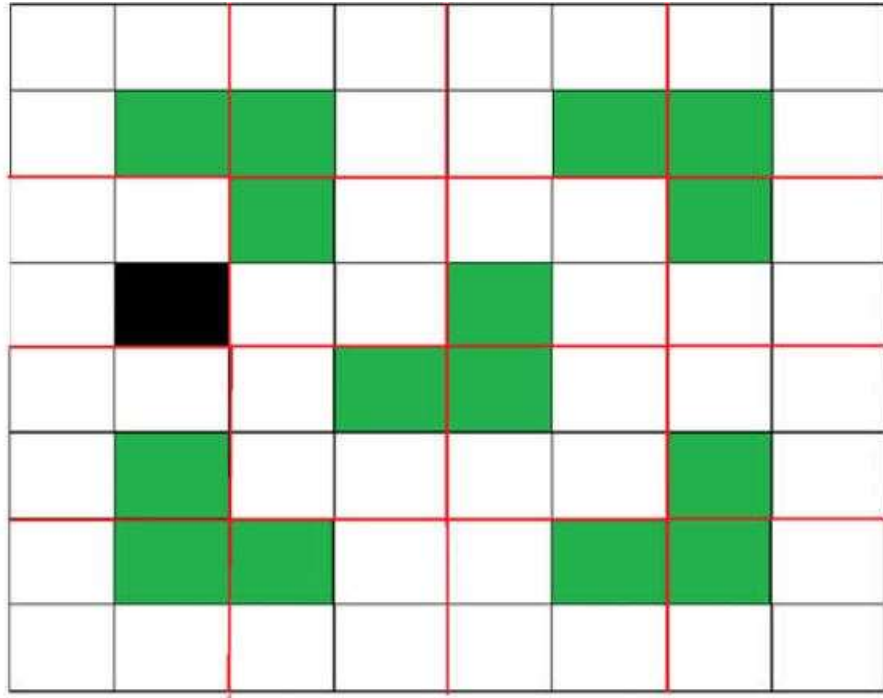


Gambar 3. Ilustrasi setelah tile pertama ditempatkan



Gambar 4 Perulangan untuk subsquare pertama.





Gambar 5. Memperlihatkan langkah pertama di keempat subsquares.

Tugas:

- Buatlah program untuk menyelesaikan problem tiling menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

/\*

Nama Program : Tiling Problem

Nama : Fadlan Mulya Priatna

NPM : 140810180041

Tanggal Pembuatan : 30 Maret 2020

Deskripsi Program : program untuk menyelesaikan problem tiling menggunakan algoritma divide & conquer yang diberikan dengan bahasa C++

\*/

```
#include <iostream>
```

```
using namespace std;
```

```
int countWays(int n, int m){
```

```
    int count[n + 1];
```

```
    count[0] = 0;
```

```
    for (int i = 1; i <= n; i++) {
```

```
        // recurrence relation
```

```
        if (i > m) {
```

```
            count[i] = count[i - 1] + count[i - m];
```

```
        }
```

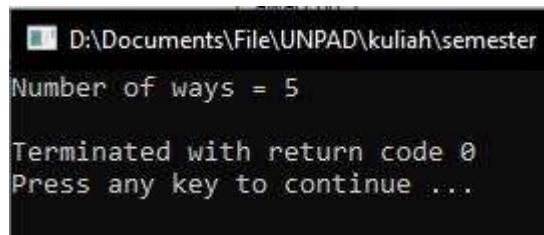
```
        else if (i < m){
```

```
            count[i] = 1;
```

```
        }
```

```
        else{
            count[i] = 2;
        }
    }
    return count[n];
}

int main(){
    int n = 4, m = 2;
    cout << "Number of ways = " << countWays(n, m);
    return 0;
}
```



- Relasi rekurensi untuk algoritma rekursif di atas dapat ditulis seperti di bawah ini.  $C$  adalah konstanta.  $T(n) = 4T(n/2) + C$ . Selesaikan rekurensi tersebut dengan Metode Master

Jawab:

Kompleksitas Waktu: Relasi perulangan untuk algoritma rekursif di atas dapat ditulis seperti di bawah ini.  $C$  adalah konstanta.  $T(n) = 4T(n/2) + C$  Rekursi di atas dapat diselesaikan dengan menggunakan Metode Master dan kompleksitas waktu adalah  $O(n^2)$ .

Cara kerjanya: Pengerjaan algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical Induction. Biarkan kuadrat input berukuran  $2k \times 2k$  di mana  $k \geq 1$ .

Kasus Dasar: Kita tahu bahwa masalahnya dapat diselesaikan untuk  $k = 1$ . Kami memiliki  $2 \times 2$  persegi dengan satu sel hilang.

Hipotesis Induksi: Biarkan masalah dapat diselesaikan untuk  $k-1$ .

Sekarang perlu dibuktikan untuk membuktikan bahwa masalah dapat diselesaikan untuk  $k$  jika dapat diselesaikan untuk  $k-1$ . Untuk  $k$ , ditempatkan ubin berbentuk L di tengah dan memiliki empat subsquare dengan dimensi  $(2k - 1) \times (2k - 1)$  seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subsquares, maka dapat menyelesaikan kuadrat lengkap.