

Bandana Irmal Abdllah  
140810180025  
Tugas 5

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

Tugas :

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

```
1.
2.  /*
3.   Nama      : Bandana Irmal Abdillah
4.   NPM       : 140810180025
5.   Closest pair of point
6.   */
7.
8. #include <bits/stdc++.h>
9. using namespace std;
10.
11. // A structure to represent a Point in 2D plane
12. class Point
13. {
14. public:
15.     int x, y;
16. };
17.
18. // Needed to sort array of points
19. // according to X coordinate
20. int compareX(const void* a, const void* b)
21. {
22.     Point *p1 = (Point *)a, *p2 = (Point *)b;
23.     return (p1->x - p2->x);
24. }
25.
26. // Needed to sort array of points according to Y coordinate
27. int compareY(const void* a, const void* b)
28. {
29.     Point *p1 = (Point *)a, *p2 = (Point *)b;
30.     return (p1->y - p2->y);
31. }
32.
33. // A utility function to find the
34. // distance between two points
35. float dist(Point p1, Point p2)
36. {
37.     return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
38.                 (p1.y - p2.y)*(p1.y - p2.y)
39.                 );
40. }
41.
42. // A Brute Force method to return the
43. // smallest distance between two points
44. // in P[] of size n
```

```

45. float bruteForce(Point P[], int n)
46. {
47.     float min = FLT_MAX;
48.     for (int i = 0; i < n; ++i)
49.         for (int j = i+1; j < n; ++j)
50.             if (dist(P[i], P[j]) < min)
51.                 min = dist(P[i], P[j]);
52.     return min;
53. }
54.
55. // A utility function to find
56. // minimum of two float values
57. float min(float x, float y)
58. {
59.     return (x < y)? x : y;
60. }
61.
62.
63. // A utility function to find the
64. // distance between the closest points of
65. // strip of given size. All points in
66. // strip[] are sorted according to
67. // y coordinate. They all have an upper
68. // bound on minimum distance as d.
69. // Note that this method seems to be
70. // a  $O(n^2)$  method, but it's a  $O(n)$ 
71. // method as the inner loop runs at most 6 times
72. float stripClosest(Point strip[], int size, float d)
73. {
74.     float min = d; // Initialize the minimum distance as d
75.
76.     qsort(strip, size, sizeof(Point), compareY);
77.
78.     // Pick all points one by one and try the next points till the difference
79.     // between y coordinates is smaller than d.
80.     // This is a proven fact that this loop runs at most 6 times
81.     for (int i = 0; i < size; ++i)
82.         for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
83.             if (dist(strip[i], strip[j]) < min)
84.                 min = dist(strip[i], strip[j]);
85.
86.     return min;
87. }
88.
89. // A recursive function to find the
90. // smallest distance. The array P contains
91. // all points sorted according to x coordinate
92. float closestUtil(Point P[], int n)
93. {
94.     // If there are 2 or 3 points, then use brute force
95.     if (n <= 3)
96.         return bruteForce(P, n);
97.
98.     // Find the middle point
99.     int mid = n/2;
100.    Point midPoint = P[mid];
101.
102.    // Consider the vertical line passing
103.    // through the middle point calculate
104.    // the smallest distance dl on left
105.    // of middle point and dr on right side

```

```

106.         float dl = closestUtil(P, mid);
107.         float dr = closestUtil(P + mid, n - mid);
108.
109.         // Find the smaller of two distances
110.         float d = min(dl, dr);
111.
112.         // Build an array strip[] that contains
113.         // points close (closer than d)
114.         // to the line passing through the middle point
115.         Point strip[n];
116.         int j = 0;
117.         for (int i = 0; i < n; i++)
118.             if (abs(P[i].x - midPoint.x) < d)
119.                 strip[j] = P[i], j++;
120.
121.         // Find the closest points in strip.
122.         // Return the minimum of d and closest
123.         // distance is strip[]
124.         return min(d, stripClosest(strip, j, d) );
125.     }
126.
127.     // The main function that finds the smallest distance
128.     // This method mainly uses closestUtil()
129.     float closest(Point P[], int n)
130.     {
131.         qsort(P, n, sizeof(Point), compareX);
132.
133.         // Use recursive function closestUtil()
134.         // to find the smallest distance
135.         return closestUtil(P, n);
136.     }
137.
138.     // Driver code
139.     int main()
140.     {
141.         Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
142.         int n = sizeof(P) / sizeof(P[0]);
143.         cout << "The smallest distance is " << closest(P, n);
144.         return 0;
145.     }

```

```

D:\CobaanHidup\Semester 4 IP 4\Analisis Algoritma\Praktikum\AnalgoKu5\ClosestPairOfPoints.exe
The smallest distance is 1.41421
-----
Process exited after 0.2056 seconds with return value 0
Press any key to continue . . .

```

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

Asumsikan bahwa kita menggunakan algoritma pengurutan  $O(n \lg n)$ . 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  $O(n \lg n)$  dan akhirnya menemukan titik terdekat dalam strip dalam waktu  $O(n)$ . Jadi  $T(n)$  dapat dinyatakan sebagai berikut :

$$T(n) = 2T(n/2) + O(n) + O(n \lg n) + O(n)$$

$$T(n) = 2T(n/2) + O(n \lg n)$$

$$T(n) = O(n \times \lg n \times \lg n)$$

Catatan :

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

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

Tugas :

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

```
2. /*
3. Nama      : Bandana Irmal Abdillah
4. NPM       : 140810180025
5. Algoritma : Karatsuba
6. */
7.
8. #include<iostream>
9. #include<stdio.h>
10.
11. using namespace std;
12.
13. int makeEqualLength(string &str1, string &str2)
14. {
15.     int len1 = str1.size();
16.     int len2 = str2.size();
17.     if (len1 < len2)
18.     {
19.         for (int i = 0 ; i < len2 - len1 ; i++)
20.             str1 = '0' + str1;
21.         return len2;
22.     }
23.     else if (len1 > len2)
24.     {
25.         for (int i = 0 ; i < len1 - len2 ; i++)
```

```

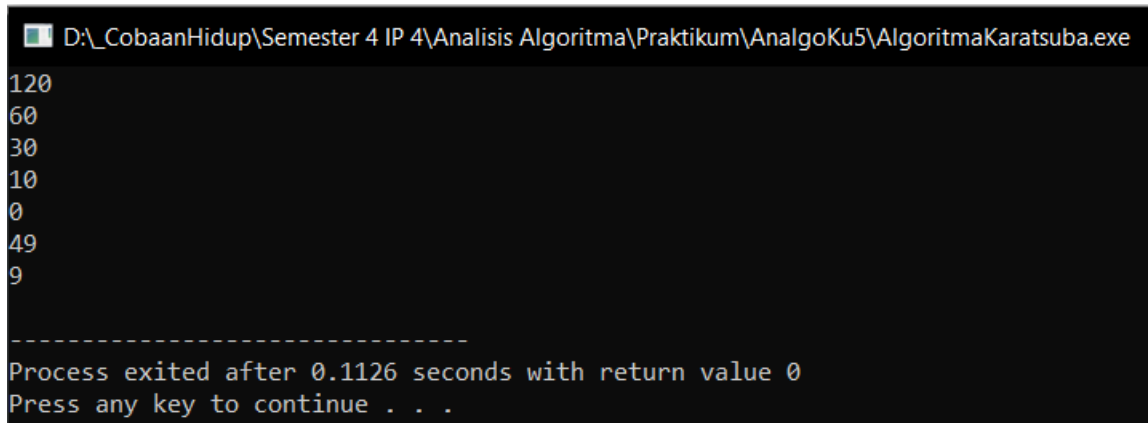
26.         str2 = '0' + str2;
27.     }
28.     return len1; // If len1 >= len2
29. }
30.
31. // The main function that adds two bit sequences and returns the addition
32. string addBitStrings( string first, string second )
33. {
34.     string result; // To store the sum bits
35.
36.     // make the lengths same before adding
37.     int length = makeEqualLength(first, second);
38.     int carry = 0; // Initialize carry
39.
40.     // Add all bits one by one
41.     for (int i = length-1 ; i >= 0 ; i--)
42.     {
43.         int firstBit = first.at(i) - '0';
44.         int secondBit = second.at(i) - '0';
45.
46.         // boolean expression for sum of 3 bits
47.         int sum = (firstBit ^ secondBit ^ carry)+'0';
48.
49.         result = (char)sum + result;
50.
51.         // boolean expression for 3-bit addition
52.         carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
53.     }
54.
55.     // if overflow, then add a leading 1
56.     if (carry) result = '1' + result;
57.
58.     return result;
59. }
60.
61. // A utility function to multiply single bits of strings a and b
62. int multiplySingleBit(string a, string b)
63. {
64.     return (a[0] - '0')*(b[0] - '0');
65. }
66.
67. // The main function that multiplies two bit strings X and Y and returns
68. // result as long integer
69. long int multiply(string X, string Y)
70. {
71.     // Find the maximum of lengths of x and Y and make length
72.     // of smaller string same as that of larger string
73.     int n = makeEqualLength(X, Y);
74.
75.     // Base cases
76.     if (n == 0) return 0;
77.     if (n == 1) return multiplySingleBit(X, Y);
78.
79.     int fh = n/2; // First half of string, floor(n/2)
80.     int sh = (n-fh); // Second half of string, ceil(n/2)
81.
82.     // Find the first half and second half of first string.
83.     // Refer http://goo.gl/1Lmgn for substr method
84.     string Xl = X.substr(0, fh);
85.     string Xr = X.substr(fh, sh);
86.

```

```

87. // Find the first half and second half of second string
88. string Yl = Y.substr(0, fh);
89. string Yr = Y.substr(fh, sh);
90.
91. // Recursively calculate the three products of inputs of size n/2
92. long int P1 = multiply(Xl, Yl);
93. long int P2 = multiply(Xr, Yr);
94. long int P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));
95.
96. // Combine the three products to get the final result.
97. return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
98. }
99.
100. // Driver program to test aboev functions
101. int main()
102. {
103.     printf ("%ld\n", multiply("1100", "1010"));
104.     printf ("%ld\n", multiply("110", "1010"));
105.     printf ("%ld\n", multiply("11", "1010"));
106.     printf ("%ld\n", multiply("1", "1010"));
107.     printf ("%ld\n", multiply("0", "1010"));
108.     printf ("%ld\n", multiply("111", "111"));
109.     printf ("%ld\n", multiply("11", "11"));
110. }

```



```

D:\_CobaanHidup\Semester 4 IP 4\Analisis Algoritma\Praktikum\AnalgoKu5\AlgoritmaKaratsuba.exe
120
60
30
10
0
49
9
-----
Process exited after 0.1126 seconds with return value 0
Press any key to continue . . .

```

2. 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 ( $n \lg n$ )

- 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  $xy = a r^n + e r^{n/2} + d$
- $T(n) = 3 T(n/2) + O(n)$
- $T(n) = O(n^{\log 3}) = O(n^{1.584...})$

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

Tugas :

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

// n adalah ukuran kotak yang diberikan, p adalah lokasi sel yang hilang  
Tile (int n, Point p)

- 1) Kasus dasar:  $n = 2$ , A  $2 \times 2$  persegi dengan satu sel yang hilang tidak ada apa-apanya tapi ubin dan bisa diisi dengan satu ubin.
- 2) Tempatkan ubin berbentuk L di tengah sehingga tidak menutupi subsquare  $n/2 \times n/2$  yang memiliki kuadrat yang hilang. Sekarang keempat subsquare ukuran  $n/2 \times n/2$  memiliki sel yang hilang (sel yang tidak perlu diisi). Lihat gambar 2 di bawah ini.
- 3) Memecahkan masalah secara rekursif untuk mengikuti empat. Biarkan p1, p2, p3 dan p4 menjadi posisi dari 4 sel yang hilang dalam 4 kotak.
  - a) Ubin ( $n/2$ , p1)
  - b) Ubin ( $n/2$ , p2)
  - c) Ubin ( $n/2$ , p3)
  - d) Ubin ( $n/2$ , p4)

2. 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.

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  $O(n^2)$

Bagaimana cara kerjanya?

Pengerjaan algoritma divide and conquer dapat dibuktikan menggunakan mathematical induction. Biarkan kuadrat input berukuran  $2k \times 2k$  dimana  $k \geq 1$ .

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, dapat menyelesaikan kuadrat lengkap.