**LAPORAN PRAKTIKUM**

**ANALISIS ALGORITMA**



**Dibuat oleh:**

AHMAD IRFAN FADHOLI

140810180034

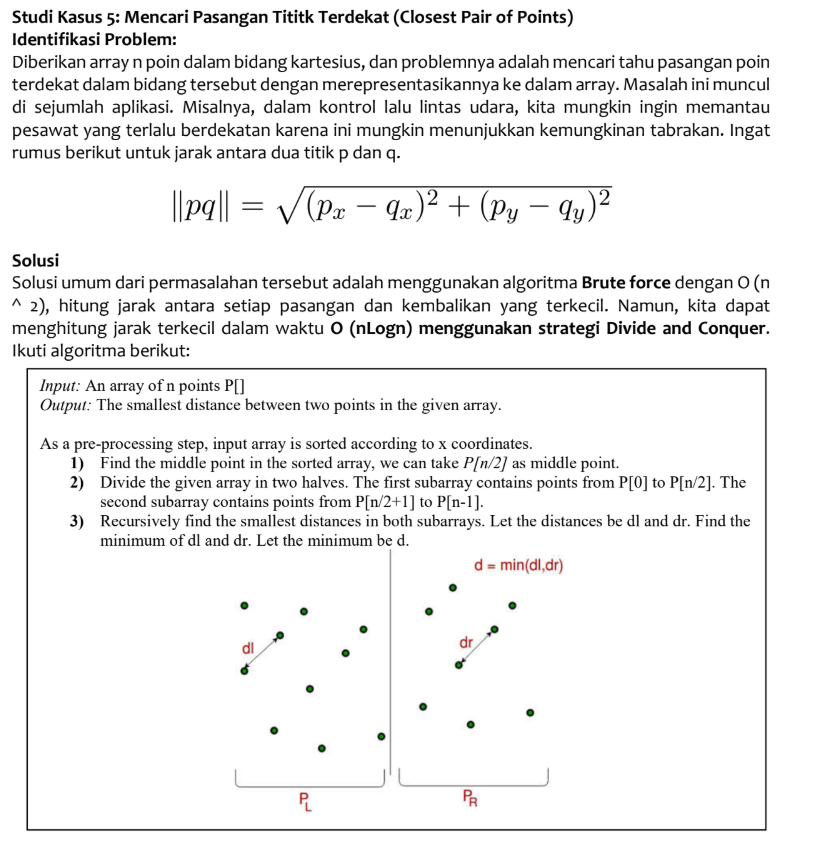
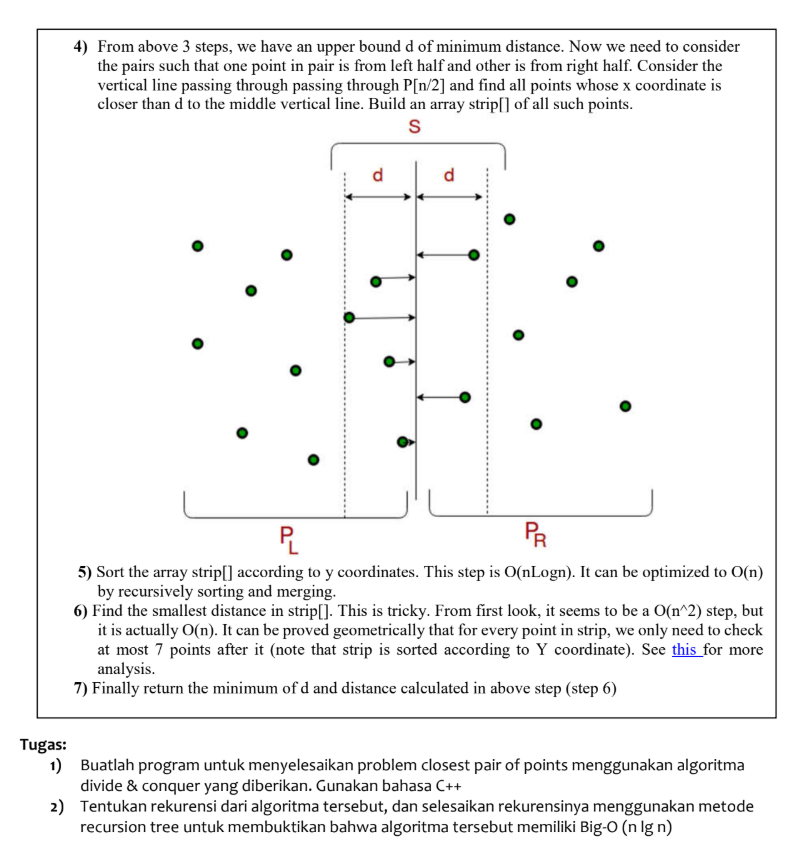
**PROGRAM STUDI S-1 TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM UNIVERSITAS PADJADJARAN**

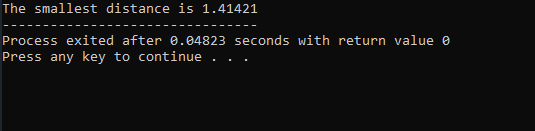
**2020**

**Studi Kasus 5**

1. Program Closest Pair of Points C++

1. /\*
2. Nama        : Ahmad Irfan Fadholi
3. Kelas       : B
4. NPM         : 140810180034
5. Deskripsi   : Closest Pair of Points
6. \*/
8. #include <iostream>
9. #include <float.h>
10. #include <stdlib.h>
11. #include <cmath>
13. **using** **namespace** std;
15. **struct** Point{
16. **int** x, y;
17. };
19. **int** compareX(**const** **void** \*a, **const** **void** \*b)
20. {
21. Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;
22. **return** (p1->x - p2->x);
23. }
25. **int** compareY(**const** **void** \*a, **const** **void** \*b){
26. Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;
27. **return** (p1->y - p2->y);
28. }
30. **float** dist(Point p1, Point p2){
31. **return** sqrt((p1.x - p2.x) \* (p1.x - p2.x) +
32. (p1.y - p2.y) \* (p1.y - p2.y));
33. }
35. **float** bruteForce(Point P[], **int** n){
36. **float** min = FLT\_MAX;
37. **for** (**int** i = 0; i < n; ++i)
38. **for** (**int** j = i + 1; j < n; ++j)
39. **if** (dist(P[i], P[j]) < min)
40. min = dist(P[i], P[j]);
41. **return** min;
42. }
44. **float** min(**float** x, **float** y){
45. **return** (x < y) ? x : y;
46. }
48. **float** stripClosest(Point strip[], **int** size, **float** d){
49. **float** min = d;
50. qsort(strip, size, **sizeof**(Point), compareY);
51. **for** (**int** i = 0; i < size; ++i)
52. **for** (**int** j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j)
53. **if** (dist(strip[i], strip[j]) < min)
54. min = dist(strip[i], strip[j]);
55. **return** min;
56. }
58. **float** closestUtil(Point P[], **int** n){
59. **if** (n <= 3)
60. **return** bruteForce(P, n);
62. **int** mid = n / 2;
63. Point midPoint = P[mid];
64. **float** dl = closestUtil(P, mid);
65. **float** dr = closestUtil(P + mid, n - mid);
66. **float** d = min(dl, dr);
67. Point strip[n];
68. **int** j = 0;
69. **for** (**int** i = 0; i < n; i++)
70. **if** (abs(P[i].x - midPoint.x) < d)
71. strip[j] = P[i], j++;
73. **return** min(d, stripClosest(strip, j, d));
74. }
76. **float** closest(Point P[], **int** n){
77. qsort(P, n, **sizeof**(Point), compareX);
79. **return** closestUtil(P, n);
80. }
82. **int** main(){
83. Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
84. **int** n = **sizeof**(P) / **sizeof**(P[0]);
85. cout << "The smallest distance is " << closest(P, n);
86. **return** 0;
87. }

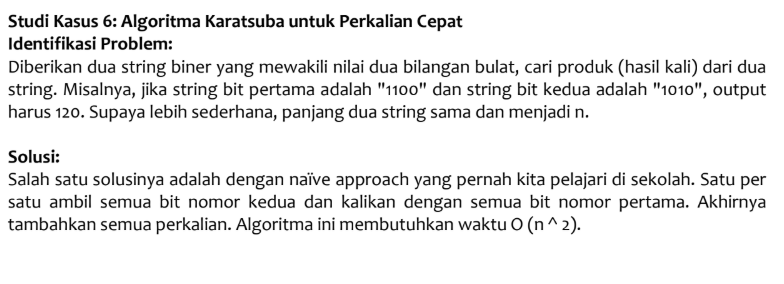
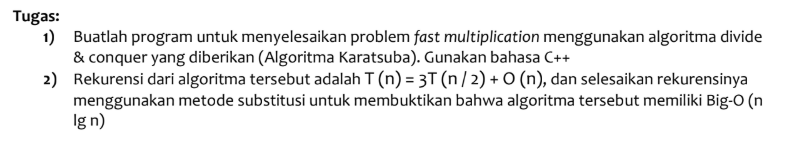


1. Kompleksitas Waktu

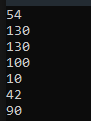
T (n) = 2T (n / 2) + O (n) + O (n Log n) + O (n)

T (n) = 2T (n / 2) + O (n Log n)

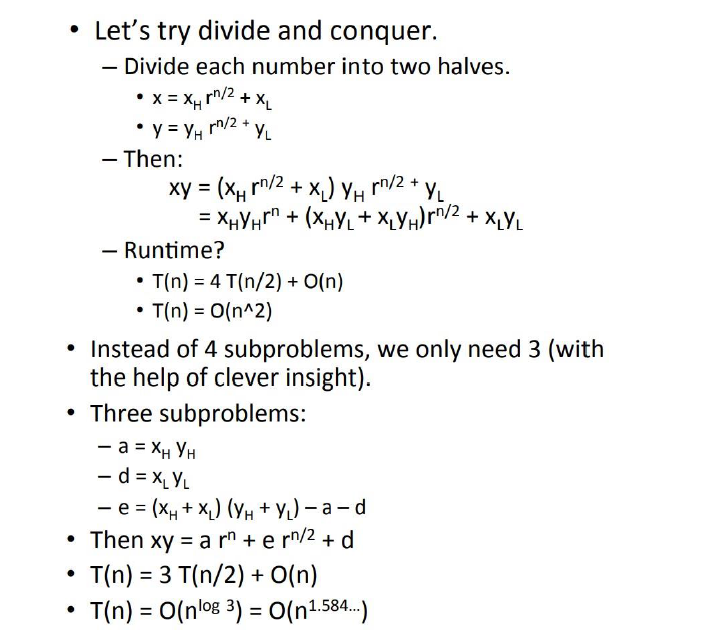
T (n) = T (n \* Log n \* Log n)

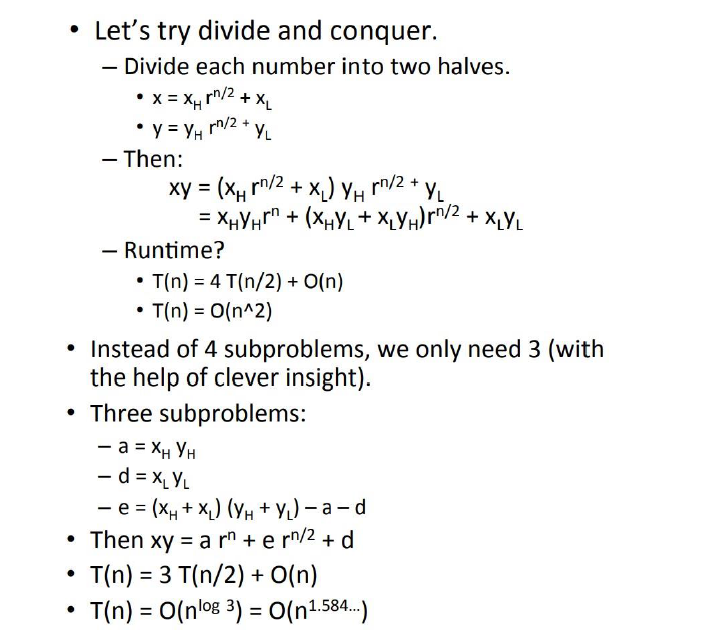
 

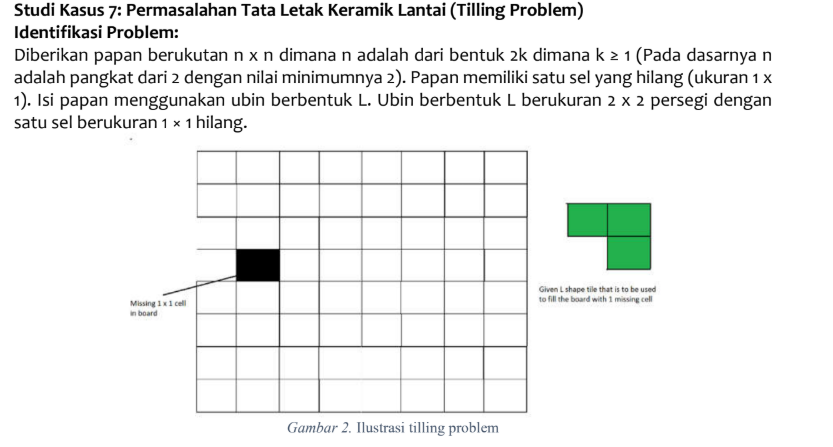
1. /\*
2. Nama        : Ahmad Irfan Fadholi
3. Kelas       : B
4. NPM         : 140810180034
5. Deskripsi   : Karatsuba Fast Multiplication Algorithm
6. \*/
7. #include<iostream>
9. **using** **namespace** std;
11. **int** makeEqualLength(string &str1, string &str2) {
12. **int** len1 = str1.size();
13. **int** len2 = str2.size();
14. **if** (len1 < len2) {
15. **for** (**int** i = 0 ; i < len2 - len1 ; i++)
16. str1 = '0' + str1;
17. **return** len2;
18. }
19. **else** **if** (len1 > len2) {
20. **for** (**int** i = 0 ; i < len1 - len2 ; i++)
21. str2 = '0' + str2;
22. }
23. **return** len1;
24. }
26. string addBitStrings( string first, string second ) {
27. string result;
28. **int** length = makeEqualLength(first, second);
29. **int** carry = 0;
30. **for** (**int** i = length-1 ; i >= 0 ; i--) {
31. **int** firstBit = first.at(i) - '0';
32. **int** secondBit = second.at(i) - '0';
33. **int** sum = (firstBit ^ secondBit ^ carry)+'0';
35. result = (**char**)sum + result;
37. carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
38. }
39. **if** (carry)
40. result = '1' + result;
41. **return** result;
42. }
44. **int** multiplyiSingleBit(string a, string b) {
45. **return** (a[0] - '0')\*(b[0] - '0'); }
47. **long** **int** multiply(string X, string Y) {
48. **int** n = makeEqualLength(X, Y);
49. **if** (n == 0)
50. **return** 0;
51. **if** (n == 1)
52. **return** multiplyiSingleBit(X, Y);
53. **int** fh = n/2;
54. **int** sh = (n-fh);
55. string Xl = X.substr(0, fh);
56. string Xr = X.substr(fh, sh);
58. string Yl = Y.substr(0, fh);
59. string Yr = Y.substr(fh, sh);
61. **long** **int** P1 = multiply(Xl, Yl);
62. **long** **int** P2 = multiply(Xr, Yr);
63. **long** **int** P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));
64. **return** P1\*(1<<(2\*sh)) + (P3 - P1 - P2)\*(1<<sh) + P2;
65. }
67. **int** main() {
68. cout<<multiply("1001", "0110")<<endl;
69. cout<<multiply("1101", "1010")<<endl;
70. cout<<multiply("1101", "1010")<<endl;
71. cout<<multiply("1010", "1010")<<endl;
72. cout<<multiply("0101", "0010")<<endl;
73. cout<<multiply("0011", "1110")<<endl;
74. cout<<multiply("1010", "1001")<<endl;
75. }

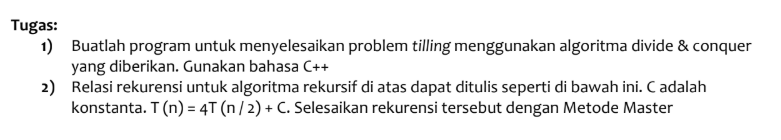


1. Komplesitas Waktu

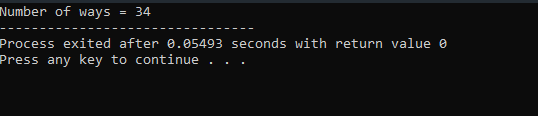








1. /\*
2. Nama        : Ahmad Irfan Fadholi
3. Kelas       : B
4. NPM         : 140810180034
5. Deskripsi   : Tilling Problem
6. \*/
7. #include <iostream>
8. **using** **namespace** std;
10. // function to count the total number of ways
11. **int** countWays(**int** n, **int** m){
12. **int** count[n + 1];
13. count[0] = 0;
14. **for** (**int** i = 1; i <= n; i++)
15. {
16. **if** (i > m)
17. count[i] = count[i - 1] + count[i - m];
18. **else** **if** (i < m)
19. count[i] = 1;
20. **else**
21. count[i] = 2;
22. }
23. **return** count[n];
24. }
26. **int** main(){
27. **int** n = 8, m = 2;
28. cout << "Number of ways = "
29. << countWays(n, m);
30. **return** 0;
31. }



1. 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 (n2)

Bagaimana cara kerjanya?

Pengerjaan algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical

Induction. Biarkan kuadrat input berukuran 2k x 2k di mana k> = 1.

Kasus Dasar: Kita tahu bahwa masalahnya dapat diselesaikan untuk k = 1. Kami memiliki 2 x 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 subsqure dengan dimensi 2k-1 x 2k-1 seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subskuares, dapat menyelesaikan kuadrat lengkap.