Penyelesaian Masalah *Closest Pair* dengan Algoritma Divide and Conquer

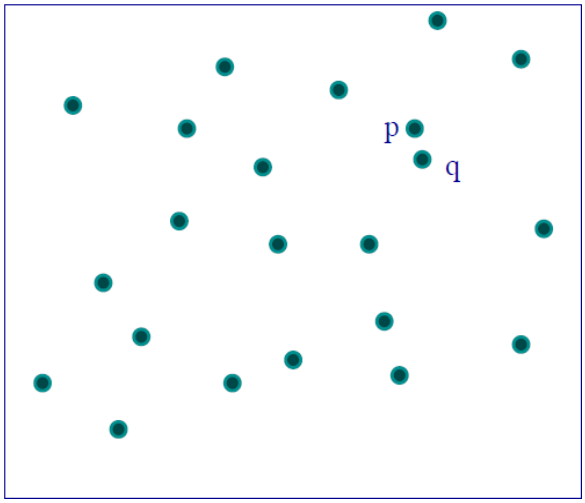
Karol Danutama 13508040
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
karoldanutama@gmail.com

Permasalahan closest pair adalah permasalahan umum yang kerap ditemui dalam konteks geometri ruang dan optimasi jarak. Dalam dunia sehari – hari juga banyak ditemui permasalahan semacam ini. Namun, seringkali kasus nyata memiliki batasan yang tinggi sehingga terkadang solusi dengan algoritma brute force tidak dapat menyelesaikan permasalahan dengan mangkus dan sangkil. Oleh sebab itu, penulis mencoba untuk mengulas salah satu bentuk optimasi solusi permasalahan ini dengan algoritma divide and conquer.

***Kata kunci**: closest pair, brute force, divide and conquer, mangkus dan sangkil*

I. PENDAHULUAN

Permasalahan *closest pair* merupakan salah satu permasalahan klasik dalam dunia matematika diskrit. Secara singkat, deskripsi permasalahan adalah sebagai berikut: diberikan N buah titik yang terletak pada bidang planar 2-dimensi, tentukanlah dua buah titik yang memiliki jarak paling dekat.



Gambar 1 Ilustrasi permasalahan *closest pair*

Secara umum, permasalahan ini sering dikaitkan dengan permasalahan pada dunia sehari – hari. Contoh permasalahannya seperti proses pengambilan dua bahan baku oleh lengan robot pada pabrik mikroprosesor agar pergerakan lengan robot seminimal mungkin.

Secara naif, permasalahan ini dapat diselesaikan dengan algoritma *brute force* yang membutuhkan kompleksitas waktu O (N^2). Untuk N yang besar (N > 100000), komputasi dengan algoritma *brute force* akan memakan waktu yang lama, bahkan dengan komputer paling cepat saat ini.

Untuk itu, penulis berusaha untuk memberikan ulasan mengenai optimasi solusi permasalahn ini dengan algoritma *divide and conquer* yang memiliki kompleksitas O (n log n). Untuk N yang besar, algoritma ini masih dirasa cukup mangkus untuk diterapkan dalam keseharian. Meskipun demikian, pencapaian algoritma *divide and conquer* membutuhkan beberapa lemma yang terbatas pada parameter tertentu.

II. PENYELESAIAN DENGAN *BRUTE FORCE*

Solusi *brute force* untuk masalah ini cukup *straightforward*: tinjau seluruh kemungkinan pasangan titik – titik kemudian hitung jaraknya dan cari nilai maksimum. *Pseudocode* untuk solusi ini adalah sebagai berikut:

```
function findClosestPair : real
min = ~
foreach i in pointList
  foreach j in pointList
    if (i <> j AND distance(i , j) < max)
      min = distance(i , j)
return min
```

Gambar 2 *Pseudocode* pencarian pasangan terpendek dengan algoritma *brute force*

Jadi, untuk setiap *instance* permasalahan, akan terjadi N *pass* untuk setiap titik dan masing – masing *pass* akan melakukan peninjauan kembali untuk N titik. Sehingga, kompleksitas waktu untuk algoritma ini adalah O(N^2).

Penulis mencoba untuk membuat dan menjalankan algoritma ini pada komputer dengan spesifikasi sebagai berikut:

- Intel Core 2 Duo 2.4 GHz

- RAM 4 GB
- FSB 800 MHz

Untuk N sama dengan 100000 dengan koordinat titik acak yang berada pada rentang -100000 hingga 100000, penulis memperoleh hasil program berjalan dalam waktu 32 detik untuk dapat mengeluarkan hasil optimum. Jika algoritma ini diterapkan dalam dunia sehari – hari, algoritma ini tetap dapat memberikan jawaban yang benar, tetapi tidak dapat berjalan secara mangkus dan sangkil.

III. PENYELESAIAN DENGAN *DIVIDE AND CONQUER*

Strategi penyelesaian dengan algoritma *divide and conquer* tidak sesederhana algoritma *brute force*. Penyelesaian dengan *divide and conquer* membutuhkan pencapaian sebuah *lemma* yang bergantung pada strategi penyelesaian itu sendiri. *Lemma* inilah yang menjadi kunci optimasi.

Pada konsepnya, penyelesaian menggunakan algoritma *divide and conquer* dapat diabstraksikan melalui *pseudocode* berikut:

```
function findClosestPair(left , right) : real
if (right = left)
  return 0
else
  l = findClosestPair(left , (left + right) /
2)
  r = findClosestPair((left + right) / 2 + 1 ,
right)
  mid = conquer(left , right)
  return min(l,r,mid)
```

Gambar 3 *Pseudocode* pencarian pasangan terpendek dengan algoritma *divide and conquer*

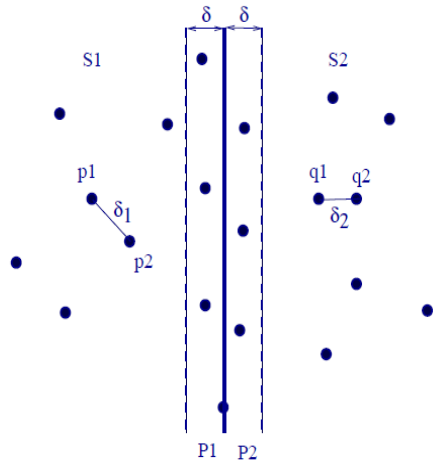
Fungsi di atas memiliki prekondisi yang harus dipenuhi, yaitu titik – titik sudah terurut berdasarkan nilai absis. Pengurutan sangat disarankan dilakukan dengan algoritma pengurutan dengan kompleksitas O (n log n) agar tidak memperburuk keseluruhan kompleksitas program.

Setelah diurutkan, titik – titik tersebut akan ditinjau untuk menghitung jarak pasangan titik terpendek. Peninjauan ini akan dilakukan satu per satu dengan membagi titik – titik menjadi dua segmen. Pembagian didasarkan pada sebuah garis vertikal yang dapat membagi titik – titik menjadi dua segmen sama besar. Segmen kiri adalah kumpulan titik – titik yang memiliki nilai absis lebih kecil atau sama dengan dari absis garis pembagi, sementara segmen kanan adalah kumpulan titik – titik yang memiliki nilai absis lebih besar atau sama dengan nilai absis garis pembagi. Untuk berikutnya, titik – titik pada segmen kiri dilambangkan sebagai himpunan *L* dan titik – titik pada segmen kanan dilambangkan sebagai himpunan *R*.

Setelah dibagi menjadi dua segmen sama besar, himpunan *L* akan ditinjau lagi dengan fungsi yang sama secara rekursif, demikian halnya dengan himpunan *R* juga akan diproses dengan cara yang sama.

Setelah hasil dari kedua pemanggilan rekursif didapatkan, fungsi akan mencari nilai terkecil dari kedua pemanggilan tersebut. Kedua pemanggilan fungsi tersebut diharapkan dapat memberikan jarak pasangan titik terpendek di kedua segmen dan secara logis dapat disimpulkan bahwa jarak terpendek untuk keseluruhan kasus adalah nilai minimal dari kedua harga tersebut.

Meskipun demikian, masih terdapat satu kemungkinan yang dapat menghasilkan jarak pasangan yang lebih pendek. Kemungkinan tersebut dapat terjadi pada pasangan dua titik yang terletak di dua segmen berbeda, atau dengan kata lain pemasangan antara titik yang berada di posisi paling kanan dari segmen kiri dengan titik yang berada di posisi paling kari dari segmen kanan. Dengan kasus seperti ini, fungsi juga harus meninjau kembali kemungkinan pasangan – pasangan yang terbentuk dari kedua segmen.



Gambar 4 Contoh kasus *closest pair* dan titik – titik yang terletak di sekitar garis pembagi

Pada fungsi di atas, proses pemasangan titik – titik yang berada di segmen kanan dengan segmen kiri dilakukan melalui fungsi **conquer**. Fungsi **conquer** ini diharapkan memberikan nilai jarak terpendek dari pasangan titik – titik segmen kiri dan segmen kanan.

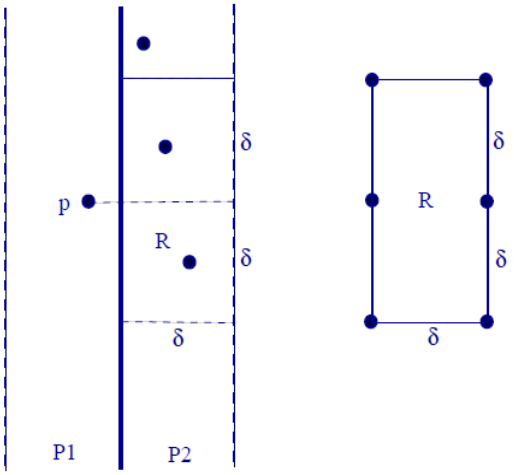
Secara naif, fungsi **conquer** akan meninjau seluruh kemungkinan pasangan antara titik – titik pada *L* dengan titik – titik pada *R*. Untuk permasalahan seperti ini, akan terdapat ½N x ½N kemungkinan pasangan. Jika algoritma ini harus melakukan pemanggilan fungsi rekursif dengan kedalaman $^2\log N$ dan pada masing – masing pemanggilan harus dilakukan iterasi sebanyak ½N x ½N kali, maka secara kasar dapat dihitung bahwa algoritma ini melakukan $¼N^2$ x $^2\log N$. Sehingga, kompleksitas waktu dari algoritma ini tidak akan membaik sebab masih berada pada rentang $O(N^2)$. Lalu, di manakah celah untuk memangkas kompleksitas algoritma ini menjadi O(N log N)?

IV. FUNGSI *CONQUER*

Jika kita melihat lebih jeli pada formulasi rekurens yang diberikan di atas, sesungguhnya untuk menghitung kemungkinan jarak terpendek yang mungkin terbentuk akibat pasangan titik yang terletak di dua pasangan berbeda, tidak diperlukan peninjauan seluruh kemungkinan titik. Peninjauan titik – titik cukup dilakukan pada titik – titik yang terletak di **sekitar** garis pembagi vertikal. Definisi **sekitar** adalah titik – titik yang terletak dalam jarak δ dari garis pembagi vertikal.

Lemma 1 : Jika δ_1 adalah jarak pasangan terpendek yang diperoleh dari himpunan *L* dan δ_2 adalah jarak terpendek yang diperoleh dari himpunan *R*, maka nilai δ adalah nilai minimum dari δ_1 dan δ_2.

Pembuktian : Jika terdapat titik A $\in$ L dan memiliki jarak ke garis lebih dari atau sama dengan δ, dan terdapat titik B $\in$ L dan memiliki jarak lebih dari atau sama dengan δ, maka jarak keduanya paling sedikit adalah 2δ. Sehingga, titik – titik yang berada di luar jarak δ, pasti tidak akan memberikan nilai optimum.



Gambar 5 Contoh kasus *closest pair* dengan *scanner* berukuran δ x 2δ

Dengan *lemma* di atas, kita dapat mengurangi jumlah titik yang harus ditinjau. Namun, pengurangan jumlah tersebut tidak dapat dihitung secara pasti sebab sangat bergantung pada kasus yang ada. Untuk mempersempit ruang pencarian, kita dapat membagi titik – titik yang berada di sekitar garis pembagi menjadi sektor – sektor. Sektor tersebut berbentuk persegi panjang yang memiliki ukuran δ x 2δ. Persegi panjang ini dapat dianggap sebagai dua buah persegi berukuran δ x δ yang dilekatkan. Sektor ini dapat dianggap sebagai sebuah *scanner* yang bergerak dari atas ke bawah untuk meninjau seluruh titik yang berada di sebelah kanan dan di sekitar garis pembagi. Pergerakan *scanner* berdasarkan pada titik – titik di sebelah kiri garis pembagi. Untuk setiap titik *p*, *scanner* akan berposisi pada ordinat *p* + δ.

Pada saat bergerak, *scanner* ini tentunya akan mencakup beberapa titik. Pada setiap konfigurasi titik yang berbebeda, kita dapat menghitung jarak terpendek antara titik *p* dengan titik – titik yang tercakup di dalam *scanner* atau jarak terpendek antara titik – titik yang berada di dalam *scanner*. Proses yang sama kemudian diulangi dengan *p* adalah titik yang berada pada sebelah kanan garis pembagi. Nilai minimum dari fungsi ini diambil dari minimum jarak terpendek dari satu kali proses *scanning*. Pada gambar ..., terlihat dua titik yang berada di dalam *scanner*.

Lemma 2 : jumlah titik yang berada di dalam *scanner* adalah 6, yaitu 6 titik yang terletak pada masing – masing titik sudut persegi.

Pembuktian : jika terdapat titik q_1 yang merupakan titik ketujuh dan berada di dalam *scanner*, maka terdapat titik q_2 yang merupakan salah satu dari 6 titik yang berada di titik sudut persegi yang memiliki jarak kurang dari δ (gambar …). Jika titik q_1 ada, maka nilai δ tidak valid. Ini merupakan kontradiksi dengan formulasi nilai δ sebelumnya.

Dengan *lemma* di atas, maka dapat disimpulkan bahwa fungsi **conquer** memiliki kompleksitas waktu $O(6^2 N) = O(N)$. Namun, pada awal fungsi **conquer**, titik – titik yang berada di sekitar garis pembagi harus terlebih dahulu diurutkan berdasarkan nilai ordinat. Jika diasumsikan pengurutan menggunakan algoritma pengurutan berorde O(N log N), maka kompleksitas total algoritma ini adalah $O(N \log^2 N)$. *Pseudocode* fungsi **conquer** adalah sebagai berikut:

```
function conquer(left , right) : real

min = ~
sortByY(left , right)

leftNodes = getLeftNodes(left , right)
rightNodes = getRightNodes(left , right)

foreach i in leftNodes
  foreach j in findMate(i , rightNodes)
    if (distance(i,j) < min)
      min = distance(i,j)

foreach i in rightNodes
  foreach j in findMate(i , leftNodes)
    if (distance(i,j) < min)
      min = distance(i,j)

return min
```

Gambar 6 *Pseudocode* fungsi **conquer**

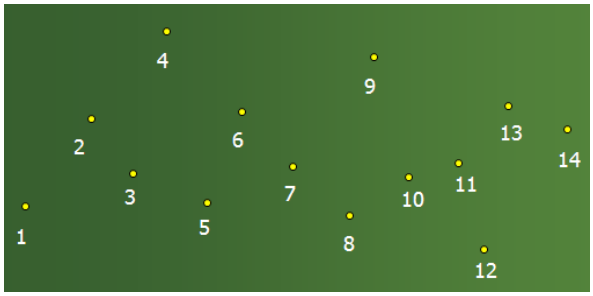
Pada fungsi di atas, terdapat tiga fungsi tambahan: **getLeftNodes**, **getRightNodes** dan **findMate**. Fungsi **getLeftNodes** dan **getRightNodes** akan mengembalikan himpunan titik – titik yang berada di sebelah kiri garis pembagi. Fungsi **findMate** akan mengembalikan titik –

titik yang menjadi kandidat pasangan jarak terpendek dengan titik yang ditinjau.

V. STUDI KASUS

Selanjutnya, penulis akan mencoba memberikan beberapa gambaran mengenai contoh kasus dari permasalahan *closest pair*. Penulis juga akan mencoba memberikan langkah – langkah penyelesaian kasus dengan algoritma *divide and conquer***.**

Jika diberikan kumpulan titik – titik sebagai berikut:

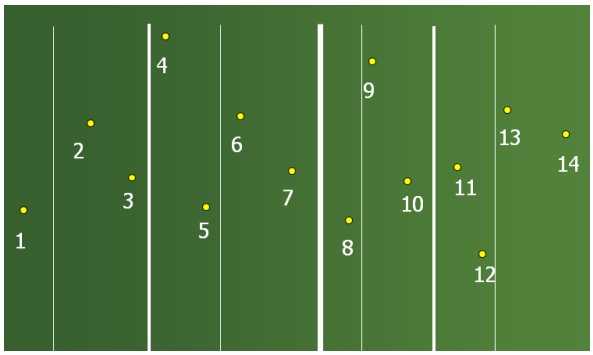


Gambar 7 Contoh kasus *closest pair* dengan 14 titik

Langkah 1: Bagi dua titik – titik menjadi dua bagian sama besar dengan garis pembagi.

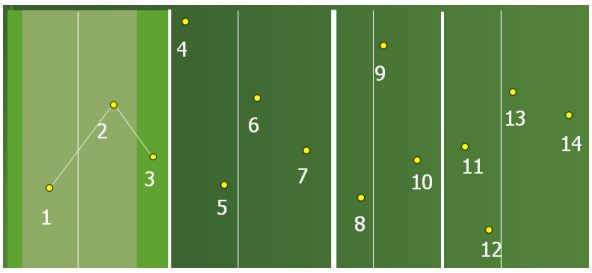
Gambar 8 Pembagian titik – titik menjadi dua segmen

Langkah 2: Bagi lagi himpunan *L* dan *R* menjadi 2 bagian sama besar, dan seterusnya secara rekursif hingga tersisa satu atau dua titik dalam satu segmen.

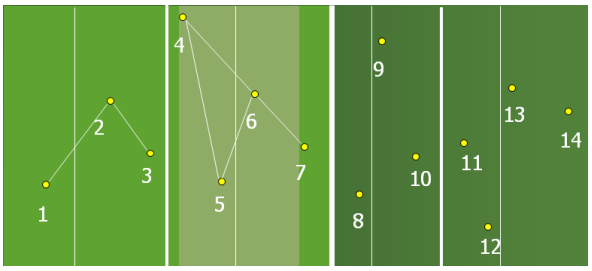


Gambar 9 Pembagian titik – titik secara rekursif

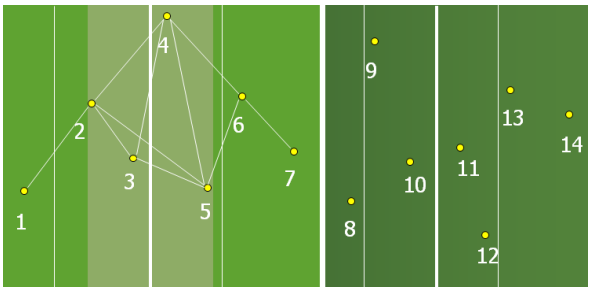
Langkah 3: Dapatkan hasil – hasil dari masing – masing segmen untuk menghasilkan hasil keseluruhan.



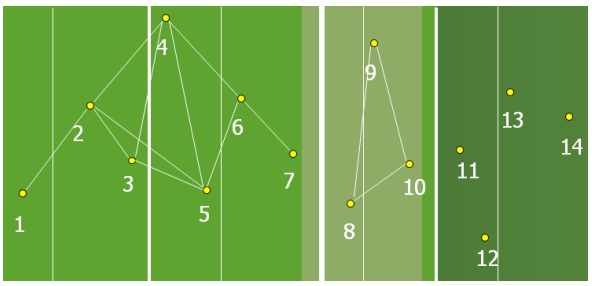
Gambar 10 Proses *conquer* untuk dua segmen terkiri



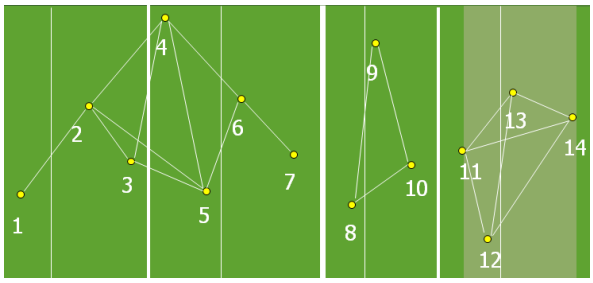
Gambar 11 Proses *conquer* untuk sebuah segmen dengan perbandingan dan area di sekitar garis pembagi



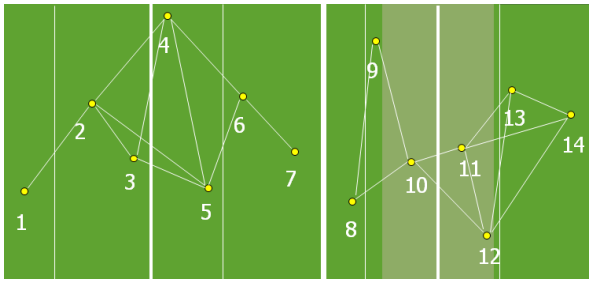
Gambar 12 Proses *conquer* untuk sebuah segmen yang merupakan gabungan dari dua segmen yang lebih kecil dengan perbandingan dan area di sekitar garis pembagi



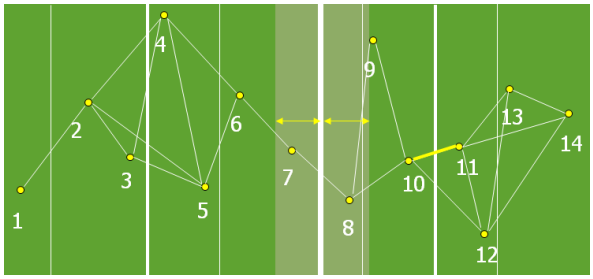
Gambar 12 Proses *conquer* untuk sebuah segmen dengan perbandingan dan area di sekitar garis pembagi



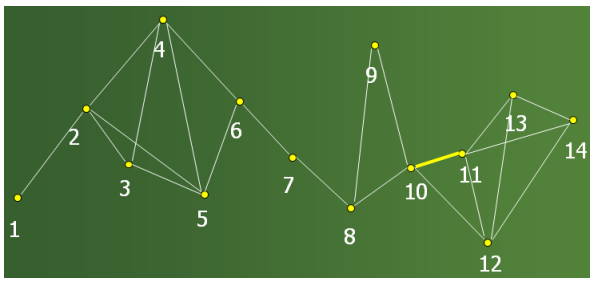
Gambar 13 Proses *conquer* untuk sebuah segmen dengan perbandingan dan area di sekitar garis pembagi



Gambar 14 Proses *conquer* untuk sebuah segmen yang merupakan gabungan dari dua segmen yang lebih kecil dengan perbandingan dan area di sekitar garis pembagi



Gambar 15 Proses *conquer* akhir dengan area di sekitar garis pembagi



Gambar 16 Hasil akhir pemecahan masalah

Dari contoh kasus di atas dengan 14 titik, terdapat 22 kali perbandingan untuk memproleh pasangan titik dengan jarak terpendek. Dari setiap langkah, jelas terlihat bahwa dalam proses *conquer* fungsi hanya perlu memeriksa titik – titik yang berada di sekitar garis pembagi. Meskipun demikian, diperlukan penelitian lebih jauh mengenai berbagai kasus *closest pair* untuk membuktikan solusi ini dengan lebih matang.

VI. KESIMPULAN

Permasalahan *closest pair* adalah permasalahan yang jamak ditemukan dalam bidang matematika diskrit geometri. Permasalahan ini pada dasarnya hanya memiliki solusi dengan algoritma *brute force*. Meskipun demikian, terdapat sebuah rancangan solusi dengan algoritma *divide and conquer* yang memiliki kemangkusa lebih baik daripada algoritma *brute force*.

Meskipun demikian, pencapaian agoritma *divide and conquer* untuk persoalan ini tidak bersifat *straightforward*, melainkan memerlukan analisis mendalam terhadap sifat – sifatk khusus yang dimiliki oleh permasalahan ini.

Oleh sebab itu, dapat disimpulkan bahwa perolehan sebuah strategi algoritma tidak harus melalui algoritma – algoritma yang sudah ada saja. Pencapaian sebuah strategi sangat bergantung pada *domain* permasalahan yang dikaji. Untuk menemukannya, memang diperlukan pemahaman lebih terhadap *domain* permasalahan yang dihadapi.

DAFTAR PUSTAKA

[1] Halim, Ph.D, Steven and Felix Halim, S.T.. *Competitive Programming. Increasing the Lower Bound of Programming Contest.* Singapore: National University of Singapore. 2010

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 7 Desember 2010

Karol Danutama / 13508040