Optimalisasi Pemilihan Komponen *Personal Computer* Menggunakan Algoritma Budgeting yang Diimplementasi dengan Dynamic Programming

Makalah IF2211 Strategi Algoritma

Joshua Christo Randiny (13517063)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung 40132, Indonesia
13517063@std.stei.itb.ac.id

***Abstract*——Komputer sudah menjadi hal biasa di kehidupan sehari-hari. Dalam membuat komputer biasanya seseorang akan dihadapkan dengan berbagai pilihan komponen. Komponen-komponen tersebut punya harga dan performa masing-masing. Dengan adanya variasi tersebut, akan sangat sulit untuk mencari kombinasi komponen yang optimal dengan budget yang terbatas. Oleh karena itu, dibutuhkan sebuah solusi untuk memudahkan dalam mencari kombinasi komponen optimal. Algoritma budgeting adalah salah satu algoritma yang dapat digunakan untuk aplikasi ini. Algoritma tersebut akan dimplementasi dengan dynamic programming. Makalah ini akan membahas tentang implementasi dan studi kasus pemilihan komponen dengan algoritma budgeting yang diimplementasi dengan dynamic programming.**

Kata Kunci—Budgeting; Dynamic programming; Komponen komputer; Pemilihan;

I. PENDAHULUAN

Sebuah *personal computer* atau komputer adalah sebuah perangkat pintar yang dapat memproses data secara cepat dan akurat. Di dunia modern ini komputer berguna untuk melakukan banyak hal seperti mencari informasi, memproses data keuangan, bermain, dan lainnya. Agar dapat berguna, sebuah komputer membutuhkan dua komponen utama yaitu perangkat lunak dan perangkat keras. Perangkat lunak adalah kode yang berjalan dan mengatur perangkat-perangkat keras yang ada untuk melakukan hal yang berguna. Akan tetapi perangkat keras juga tidak kalah penting, tanpa perangkat keras, sebuah komputer tidak bisa melakukan apa pun. Perangkat keras komputer terdiri dari komponen-komponen yang saling bekerja sama, di antaranya adalah CPU(*Central Processing Unit*), GPU (*Graphic Processing Unit)*, dan media penyimpanan. Kedua komponen tersebut merupakan tiga komponen utama dalam sebuah komputer yang menentukan performanya dan biasanya paling mahal dibanding komponen lainnya.

Masalah yang muncul kemudian adalah dalam memilih komponen-komponen tersebut. Model yang disediakan oleh perusahaan-perusahaan komputer sangatlah beragam dengan harga dan performa berbeda-beda. Ada komponen yang mahal tapi performanya kurang dibandingkan model lain yang lebih murah. Kadang ada juga komponen dengan harga lebih mahal dan lebih bagus performanya, akan tetapi tidak sebanding dengan harganya. Biasa perusahaan penjual komputer jadi, seperti Dell dan HP, akan mengambil lagi keuntungan dari harga komponen. Untuk itu, lebih baik untuk memilih dan membeli sendiri komponen-komponen yang akan digunakan. Akan tetapi harus hati-hati, karena jika salah memilih, hasil komputer tidak akan maksimal dan uang terbuang. Mengingat harga komputer yang bisa mencapai lebih dari 10 juta rupiah, maka hal ini harus sangat diperhatikan. Namun melihat dan mencari secara manual tidaklah mudah karena banyaknya kombinasi yang mungkin. Untuk itu diperlukan sebuah cara untuk membantu proses pemilihan ini

Optimalisasi proses pemilihan tersebut bisa dilakukan dengan menggunakan cara *brute force* untuk semua kombinasi, tapi akan lama dan berat. Oleh karena itu, metode *dynamic programming* dapat digunakan untuk mendapatkannya. Algoritma akan memilih satu komponen pada tiap kategori dengan batasan harga yang diinginkan dengan mempertimbangkan juga performa dari komponen tersebut.

Gambar 1. Beberapa dari banyak GPU yang tersedia untuk dipilih. Sumber: https://www.tomshardware.com/reviews/gpu-hierarchy,4388.html

II. DASAR TEORI

A. *Dynamic Programming*

Program dinamis atau d*ynamic programming* adalah salah satu metode pemecahan masalah. Ciri dari metode ini adalah adanya pemecahan masalah menjadi sekumpulan tahap yang lebih sederhana dan tiap tahapnya berhubungan dengan tahap sebelumnya. Ciri lain dari metode program dinamis adalah dengan adanya tabel yang berguna agar jawaban tahap sebelumnya tidak dihitung berkali-kali, melainkan hanya sekali saat mengisi tabel. Program dinamis kadang disebut juga mirip dengan metode rekursif yang dibuat tidak rekursif.

Istilah penting pada program dinamis adalah tahap dan status. Tahap adalah representasi suatu bagian pemikiran. Tiap tahap diselesaikan dengan metode biasa. Sedangkan status adalah informasi yang terasosiasi dengan tiap tahapnya. Status harus dapat menyampaikan cukup informasi untuk membuat keputusan di masa depan.

Salah satu prinsip yang sangat dijunjung di program dinamis adalah prinsip optimalitas. Jika sebuah solusi total optimal, maka setiap bagian penyusunnya pada tiap tahap juga optimal. Misalkan sekarang ada pada tahap k, artinya hasil optimal dari tahap k-1 akan digunakan dan digabungkan dengan pilihan paling optimal sekarang. Demikian juga untuk *cost* yang akan diakumulasi dari tahap-tahap sebelumnya

Karakteristik dari persoalan dengan program dinamis adalah

- Pilihan yang ada berhingga (ada batasnya) dan harus dipilih. Pada tiap tahapnya hanya bisa diambil satu keputusan atau pilihan.
- Solusi di tiap tahap dibangun atas solusi di tahap sebelumnya (mirip seperti rekursif yang solusinya dibangun atas input dari pemanggilan sebelumnya).
- *Cost* yang meningkat secara teratur pada tiap tahap. *Cost* merupakan akumulasi *cost* sebelumnya dan *cost* pada tahap tersebut.
- Prinsip optimalitas berlaku pada persoalan.
- Dapat menghasilkan lebih dari satu rangkaian hasil

Program dinamis dapat didekati dari dua arah, maju dan mundur. Program dinamis maju artinya program tersebut akan mulai dari tahap 1 hingga tahap akhir. Sedangkan untuk mundur adalah kebalikannya, dari tahap akhir ke tahap awal. Dalam mengembangkan algoritma program dinamis, ada langkah-langkah yang dapat diikuti

- Karakteristikkan struktur solusi optimal
- Definisikan secara rekursif solusi
- Hitung nilai solusi tanpa rekursif
- Konstruksi solusi optimal

Alasan mengapa program dinamis lebih cepat dari rekursif disebabkan cara rekursif bekerja. Pada rekursif, saat tiap kali masuk ke fungsi sendiri maka akan membuat variabel baru dan bisa jadi ada hal sama yang dihitung lagi. Berbeda dengan program dinamis yang menyimpan semua hasil kalkulasi untuk dapat digunakan kembali. Kekurangan dari program dinamis mungkin terletak pada kesulitannya dibanding strategi-strategi lain.

Contoh sederhana dari program dinamis adalah proses menghitung bilangan Fibonacci. Satu angka pada barisan Fibonacci terdefinisi dari penambahan dua angka sebelumnya. Secara sekilas, mungkin akan terlihat mudah jika menggunakan rekursif. Berikut ada kode untuk menghitung bilangan Fibonacci dengan rekursif dan program dinamis

Kode Rekursif	Program Dinamis

function fib(n:**integer**)
  if n <= 1 **then**
  -> n
  else
  -> fib(n-1) + fib(n-2)
	function fib(n:**integer**)
	  f : **array of integer**
	  f[0] = 0
	  f[1] = 1
	
	  i **traversal** [2..i]
	  f[i] = f[i-1] + f[i-2]
	
	  -> f[n]

Kode 1. Ilustrasi Kode Rekursif dan Program Dinamis

Untuk memudahkan, misalkan kedua fungsi tersebut dipanggil dengan parameter n = 3. Pada rekursif yang harus dilakukan adalah memanggil fungsinya sendiri untuk n = 2 dan n = 1. Saat menjalankan untuk n = 2, harus memanggil lagi fungsi untuk n = 1 dan n = 0. Hal tersebut tidaklah efisien untuk jumlah banyak. Berbeda dengan program dinamis yang akan menyimpan semua hasil kalkulasi sehingga hanya perlu menghitung sekali untuk tiap elemen.

B. *Budgeting*

Budgeting adalah suatu proses yang dilakukan untuk mengatur keuangan dengan sebaik mungkin agar hasilnya optimal. Pada *budgeting* akan ada beberapa proyek atau pekerjaan yang masing-masing punya *cost* dan *value*. Setelah *budgeting* sukses, dari tiap tahap akan terpilih satu yang mempunyai *value* maksimal untuk maksimum *cost* tertentu. Algoritma ini biasa dipakai di perusahaan-perusahaan besar untuk menentukan keputusan pada tingkat-tingkat atas.

Algoritma *budgeting* ini memakan waktu cukup lama karena harus menguji banyak kemungkinan. Oleh karena itu biasa diimplementasikan menggunakan metode program dinamis untuk mempercepat operasi.

C. *Budgeting Menggunakan Program Dinamis*

Seperti sudah dibahas sebelumnya, penggunaan program dinamis dapat mempercepat perhitungan untuk algoritma *budgeting*. Misalkan ada sebuah perusahaan yang mempunyai 2 pabrik. Tiap pabrik dapat melakukan satu macam produk selama jangka waktu tertentu. Tiap produk membutuhkan biaya yang berbeda untuk menyiapkan peralatan dan karyawan. Tiap produk juga memiliki ekspektasi keuntungan berbeda-

beda. Semakin tinggi biaya investasi, ekspektasi keuntungan akan meningkat juga. Untuk memperjelas, dapat melihat tabel di bawah ini.

Tabel 1 - Contoh Data Simulasi Sebuah Pabrik

Pabrik	Produk	Biaya	Ekspektasi Keuntungan
A	Prosesor	2	5
	RAM	1	3
B	Ponsel	3	7
	Tablet	4	9

Algoritma ini bekerja dengan pertama membagi persoalan ke tahap-tahapan. Tahapan yang akan dipakai adalah tahapan dalam memberikan dana. Karena ada 3 pabrik, maka akan terdapat 3 tahap. Status dari tiap tahap akan menyatakan jumlah modal yang dikeluarkan di tiap tahapnya. Misalkan simbol-simbol berikut ini

- k : tahap
- x_k : Status pada tahap ke *k*
- p_k : Pilihan pada tahap ke *k*
- $c_k(p_k)$: Biaya pada tahap ke *k* jika memilih p_k
- $R_k(p_k)$: Keuntungan saat memilih p_k di tahap *k*
- $f_k(x_k)$: Keuntungan optimal dari tahap awal hingga *k*

Relasi rekurens akan menjadi seperti

$$f_1(x_1) = \max \{ R_1(p_1) \}$$

$$f_k(x_k) = \max \{ R_k(p_k) + f_{k-1}[x_k - c_k(p_k)] \}$$

Setelah itu akan dimulai proses menghitung per tahap dengan asumsi diberikan *budget* sebesar 5 satuan.

Tabel 2 - Contoh Pengerjaan Budgeting (Tahap 1)

x_1	$R_1(p_1)$		Solusi Optimal	
	p_1 = Prosesor	p_1= RAM	$f_1(x_1)$	p_1
0	0	0	0	-
1	0	3	3	RAM
2	5	3	5	Prosesor
3	5	3	5	Prosesor
4	5	3	5	Prosesor
5	5	3	5	Prosesor

Nilai dari $R_k(p_k)$ akan diisi dengan keuntungan jika *cost* dari pilihan tersebut lebih kecil atau sama dengan status *budget* yang diberikan (x_1). Dari perhitungan didapat bahwa agar solusi optimal, dapat membuat prosesor atau RAM tergantung pada *budget* yang tersedia. Jika tersisa *budget* 2 satuan, lebih optimal prosesor, tapi jika hanya tersisa 1 satuan, lebih optimal RAM. Jika tidak tersisa apa-apa, tidak ada yang bisa dibuat dan dikosongkan saja.

Demikian juga dilakukan untuk tahap kedua. Perbedaannya adalah, karena ini tahap terakhir, nilai x_2 yang dipedulikan hanya yang maksimal karena agar *budget* terpakai semua.

Tabel 3 - Contoh Pengerjaan Budgeting (Tahap 2)

x_2	$R_2(p_2) + f_1[x_1 - c_1(p_1)]$		Solusi Optimal	
	p_2 = Ponsel	p_2= Tablet	$f2(x_2)$	p_2
5	7 + 5 = 12	9 + 3 = 12	12	Ponsel atau tablet

Misal di kolom p_2 = Ponsel, ponsel memakan biaya sebesar 3 sehingga tersisa biaya 2 yang jika dilihat di tabel 1 menghasilkan optimum prosesor dengan keuntungan 5. Keuntungan tersebut ditambahkan ke keuntungan ponsel sebesar 7, menjadi 12.

Setelah itu, akan dirunut ke belakang tiap-tiap pilihan untuk mendapat solusinya, di mulai dari tahap paling akhir.

Tabel 4 - Contoh Pengerjaan Budgeting (Solusi)

x_2	p_2	x_1	p_1	p_1, p_2
5	Ponsel	5 – 3 = 2	Prosesor	Ponsel dan prosesor
	Tablet	5 – 4 = 1	RAM	Tablet dan RAM

III. Analisis Masalah dan Implementasi Algoritma

A. Studi Kasus Permasalahan Pemilihan Komponen Komputer

Pemilihan komponen komputer merupakan salah satu masalah yang sebenarnya dapat dimodelkan dengan model seperti *budgeting*. Tiap pabrik dalam contoh *budgeting* di atas adalah tiap komponen yang harus ada pada sebuah komputer. Sama seperti pabrik yang hanya bisa memilih 1 produk untuk diproduksi, satu komputer umumnya hanya bisa memiliki 1 komponen dengan tipe sama. Jarang sekali ada komputer dengan lebih dari 1 CPU.

Tiap produk pada contoh pabrik di atas akan dimodelkan menjadi pilihan komponen dengan tipe tertentu. Saat sebuah komponen dipilih, akan ada biaya yang harus dikeluarkan, sama seperti contoh pabrik di atas. Biaya sebuah komponen adalah biaya untuk membeli komponen tersebut. Sedangkan untuk ekspektasi keuntungan, beranalogi dengan performa

suatu komponen. Performa ini dapat diukur dengan perangkat lunak khusus yang terstandardisasi biasa disebut juga *benchmark*. Nilai performa tersebut yang akan menjadi *value* dari suatu komponen.

Selain itu, *value* suatu komponen bisa dibuat relatif terhadap prioritas. Jika seseorang ingin komputer dengan kekuatan grafik tinggi, maka skala performa GPU bisa dikali dengan sebuah *modifier* tertentu. Dapat juga kebalikannya, jika seseorang tidak peduli dengan performa *disk*, nilai performa SSD bisa dikecilkan dengan dikalikan *multiplier* lebih kecil dari nol.

Tujuan *budgeting* komponen komputer juga sama, yaitu untuk memaksimalkan *value* dalam hal ini performa dengan *budget* atau *cost* tertentu. Nilai *budget* ini akan ditentukan oleh pengguna yang ingin membeli komponen tersebut.

Gambar 2. Ilustrasi contoh-contoh komponen komputer.
Sumber: http://www.a1-electronics.net/a-note-about-gaming-ready-pc-components/

Sebagai studi kasus penulis akan memodelkan permasalahan pemilihan komponen komputer ini dengan sebuah skenario. Seorang pelajar ingin membeli komputer baru agar dapat bermain di sela-sela waktu luang. Namun karena statusnya sebagai pelajar, pelajar tersebut hanya memiliki *budget* maksimal sebesar 1000 USD atau sekitar 15 juta rupiah.

Untuk membatasi pencarian, komponen yang bisa dibeli adalah komponen nyata dengan harga yang diambil dari situs belanja secara daring. Data-data untuk performa akan diambil dari beberapa web yang menyediakan *benchmark* terstandardisasi. Komponen yang akan dimodelkan adalah CPU (*Cental Processing Unit*), GPU (*Graphic Processing Unit*), dan SSD (*Solid State Drive*). Komponen tersebut dipilih karena perannya yang penting dan persentase total dalam harga komputer.

Tabel 5 - Data Beberapa CPU yang Sedang Banyak Dibeli

Nama	Harga (USD)	Performa
AMD Ryzen TR 2990WX	1900	1300
Intel Core i9-9980XE	2300	1250
AMD Ryzen TR 1950X	550	1060
Intel Core i5-9600K	265	1040
Intel Core i7-8700	310	1020
AMD Ryzen 7 2700X	300	998
Intel Core i7-7700K	360	964
Intel Core i5-8600	270	961
AMD Ryzen 5 2600X	180	930
AMD Ryzen 7 1700X	170	897
Intel Core i7-4770K	130	786
AMD Ryzen 5 2500X	120	783

Sumber : https://userbenchmark.com/page/developer

Tabel 6 - Data Beberapa GPU yang Sedang Banyak Dibeli

Nama	Harga (USD)	Performa
Nvidia RTX 2080 Ti	1300	1990
Nvidia GTX 1080-Ti	1200	1510
Nvidia RTX 2080	800	1480
AMD Radeon VII	700	1430
Nvidia RTX 2070	600	1310
AMD RX Vega 64-LC (Liquid Cooled)	500	1300
Nvidia GTX 1080	550	1210
Nvidia GTX 1070-Ti	450	1140
Asus GTX 1070 Ti 8GB STRIX	500	1140
AMD RX Vega 56	400	1090

Sumber : https://userbenchmark.com/page/developer

Tabel 7 - Data Beberapa SSD yang Sedang Banyak Dibeli

Nama	Harga (USD)	Performa
Intel 900P	500	421
Samsung 970 Pro	160	338
Samsung 970 Evo Plus	130	291
Samsung 960 Pro	200	281
Samsung 970 Evo	130	270

Sumber : https://userbenchmark.com/page/developer

B. Langkah-langkah penyelesaian

Untuk menyeragamkan, variabel yang akan dipakai sama dengan pada contoh pabrik. Selain itu akan ada beberapa asumsi dan batasan yang dapat dibuat

- Harga komponen total tidak boleh lebih dari *budget*. Jika kurang diperbolehkan
- Asumsi skala nilai performa sudah sesuai dengan prioritas pembeli
- Asumsi semua produk dapat dibeli dengan harga MSRP, tidak ada perubahan harga dari data yang sudah diinput
- Asumsi komponen lain pada sistem tidak dipedulikan karena tidak memakan biaya besar
- Asumsi tidak ada kebutuhan khusus pada sistem, yang membutuhkan merek atau model tertentu yang harus dipenuhi

Dengan asumsi-asumsi tersebut, program dapat dijalankan. Algoritma dari program ini adalah sebagai berikut

1. Untuk setiap tipe komponen yang artinya untuk tiap tahap (k)
    a. Siapkan $f_k(x_k)$ dan p_k diisi tanda invalid (-1) untuk membedakan kosong atau terisi yang angkanya -1
    b. Untuk setiap kemungkinan *budget* dan pilihan item p_k
        i. Kalkulasi *cost* $c_k(p_k)$ untuk tiap pilihan di tahap tersebut
        ii. Pilihan p_k yang masuk ke kriteria harga akan dicatat untuk dihitung *value* $R_k(p_k)$
        iii. Jika bukan tahap awal (p_1), tambahkan hasil optimum dari tahap sebelumnya dengan sisi *budget*(budget dikurangi *cost* pilihan di tahap tersebut) menjadi *value* di tahap sekarang
        iv. Setelah semua kemungkinan sudah dicek, cari nilai *value* terbesar dan catat untuk dipakai berikutnya
2. Untuk tiap hasil akhir di tahap akhir
    a. Telusuri pilihan p_k hingga mencapai tahap awal menggunakan DFS. Tahap awal adalah tahap pemilihan komponen pertama (k=1)
    b. Saat mencapai tahap awal, telusuri langkah untuk mendapat kemungkinan komponen
    c. Output hasil tersebut
    d. Dapatkan nama dari komponen yang dioutput

Kode sumber dari program yang dibuat dapat dilihat melalui lampiran yang penulis lampirkan di Apendiks.

C. Hasil

Dari percobaan dapat dilihat hasil berikut

```
PS C:\Users\joshu\Desktop\temp-makalah\makalah> python3 budgeting.py 1000
{'name': 'Intel Core i5-9600K', 'cost': 265, 'value': 1040}
{'name': 'AMD RX Vega 64-LC (Liquid Cooled)', 'cost': 500, 'value': 1300}
{'name': 'Samsung 970 Pro NVMe PCIe M.2 1TB', 'cost': 160, 'value': 338}
PS C:\Users\joshu\Desktop\temp-makalah\makalah>
```

Gambar 3. Hasil Output Dari Algoritma Dengan Input *Budget* 1000

Jadi menurut data yang dimiliki dan algoritma yang digunakan, untuk *budget* 1000 kombinasi komponen terbaik adalah

- Intel Core i5-9600K
- AMD RX Vega 64-LC
- Samsung 970 Pro

Untuk performa, operasi tersebut berhasil dijalankan dalam waktu 2 mili detik. Hasil yang cukup cepat. Berikutnya dibuat data *dummy* sebanyak 5000 untuk tiap kategori CPU dan GPU. Dengan data sebanyak itu, *query* dijalankan dalam waktu 1 detik.

```
PS C:\Users\joshu\Desktop\temp-makalah\makalah> python3 budgeting.py 700
Process time : 1.1861686706542969 ms
{'name': 'Test', 'cost': 142, 'value': 1371}
{'name': 'test gpu', 'cost': 420, 'value': 2000}
{'name': 'Samsung 970 Evo Plus NVMe PCIe M.2 500GB', 'cost': 130, 'value': 291}
{'name': 'Test', 'cost': 142, 'value': 1371}
{'name': 'test gpu', 'cost': 404, 'value': 2000}
{'name': 'Samsung 970 Evo Plus NVMe PCIe M.2 500GB', 'cost': 130, 'value': 291}
```

Gambar 4. Hasil Output dari Algoritma dengan 10000 Data Dummy

Jika dilihat secara manual, hasil ini terlihat baik, karena jika dijumlahkan total harga mencapai 925, sangat mendekati 1000. Jika terdapat komponen yang lebih banyak, mungkin angka tersebut bisa lebih mendekati 1000. Namun, pertanyaannya sekarang, dengan dapat mengetahui hal ini, dapat dilihat konfigurasi komputer terbaik di tiap harga.

Tabel 8 - Hasil Eksperimen Budget

Budget	Hasil	Harga (USD)	Performa
700	AMD Ryzen 7 1700X	170	897
	AMD RX Vega 56	400	1090
	Samsung 970 Evo Plus	130	291
800	AMD Ryzen 7 1700X	170	897
	AMD RX Vega 64-LC	500	1300
	Samsung 970 Evo Plus	130	291
900	Intel Core i5-9600	265	1040
	AMD RX Vega 64-LC	500	1300
	Samsung 970 Evo Plus	130	291
1000	Intel Core i5-9600	265	1040

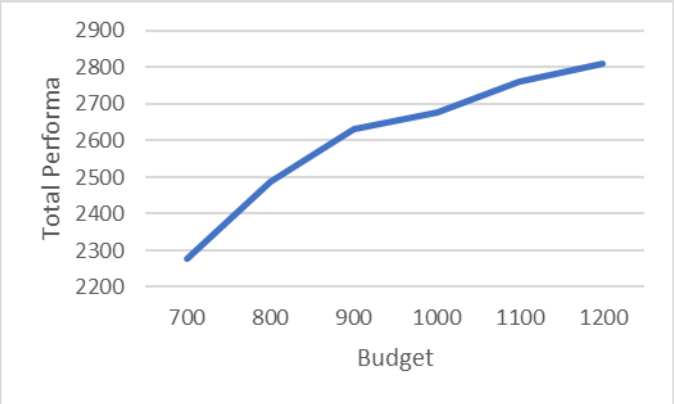
Budget	Hasil	Harga (USD)	Performa
	AMD RX Vega 64-LC	500	1300
	Samsung 970 Pro	160	338
	Intel Core i5-9600	265	1040
	AMD Radeon VII	700	1430
1100	Samsung 970 Evo Plus	130	291
	Intel Core i5-9600	265	1040
	Nvidia RTX 2080	800	1480
1200	Samsung 970 Evo Plus	130	291

D. Analisis

Dari data eksperimen, ada bagian yang menarik, yaitu pada tabel 8 data *budget* 1000 dan 1100. Pada *budget* 1000 SSD yang digunakan lebih tinggi performanya dibanding SSD di *budget* 1100. Akan tetapi nilai itu di-*offset* dengan performa grafis yang jauh lebih tinggi sehingga nilai total performa tetap lebih tinggi. Hal ini membuktikan bahwa algoritma berhasil mendapatkan nilai performa optimal.

Performa operasi juga berjalan dengan baik, untuk 10000 data yang cukup merepresentasikan jumlah komponen pada dunia nyata, algoritma berhasil menemukan kecocokan dalam waktu kurang lebih 1 detik.

Selain itu jika dilihat perbandingan antara harga dan total performa, dapat terlihat jelas peningkatannya seiring dengan kenaikan harga.



Gambar 4. Bagan Total Performa Dibanding Budget

Akan tetapi yang menarik lagi adalah adanya penurunan rasio antara performa dan harga. Semakin tinggi harga komponen, performa juga semakin tinggi, tapi tidak secepat kenaikan harganya.

Gambar 5. Bagan Performa/Harga Dibanding Budget

IV. KESIMPULAN

Dari hasil pembahasan dan eksperimen dapat disimpulkan bahwa

- Pemilihan komponen komputer dapat dioptimalisasi dengan menggunakan algoritma *budgeting*
- Algoritma *budgeting* dapat berjalan dengan baik dan mendapatkan komponen dengan performa terbaik untuk *budget* tertentu
- Implementasi program dinamis untuk algoritma berjalan dengan cukup baik dilihat dari hasil yang cukup cepat

V. APPENDIKS

Implementasi program yang dibuat penulis beserta data uji yang digunakan untuk membuat makalah ini dapat diakses pada GitHub penulis melalui tautan https://github.com/jrandiny/PC_Budget. Program ditulis dengan bahasa Python3 sehingga seharusnya kompatibel dengan semua implementasi Python3. Keterangan lebih lanjut untuk menjalankan terdapat pada readme di tautan tersebut.

UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas segala pertolongannya yang menyanggupkan penulis untuk menyelesaikan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada Bapak Rinaldi Munir sebagai pengajar mata kuliah Strategi Algoritma pada kelas K03 selama semester ini. Kepada keluarga dan teman yang telah membantu dari berbagai bidang baik langsung maupun tidak langsung, penulis ucapkan terima kasih

REFERENSI

[1] R. Munir. Diktat Kuliah IF2211 Strategi Algortima. Program Studi Teknik Informatika ITB. 2018.

[2] S.P. Bradley, A.C. Hax, and T.L. Magnanti, Applied Mathematical Programming, Addison-Wesley Publishing. 1977, pp 320-327

[3] https://www.geeksforgeeks.org/dynamic-programming/ terakhir diiakses pada 26 April 2019

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, 26 April 2019

Joshua Christo Randiny
13517063