Analisis Algoritma Penyortiran Bogosort dan Bozosort dengan Kombinatorika dan Kompleksitas Algoritma

Jonathan Levi - 13523132[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*jonathanlevi12345678@gmail.com*, *13523132@std.stei.itb.ac.id*

Abstract*—Algoritma penyortiran (*sorting algorithm*) merupakan algoritma yang berperan penting dalam menyortir atau mengurutkan deretan angka dan/atau huruf. Ada berbagai macam algoritma penyortiran yang sering digunakan, misalnya *selection sort, bubble sort, Timsort, heap sort,* dan *quick sort*. Namun, ada juga algoritma penyortiran yang sangat buruk dan hanya bergantung pada keberuntungan (waktu penyelesaian sangat inkonsisten), seperti *bogosort* dan variasinya, yaitu *bozosort.

***Keywords*—Algoritma penyortiran, *bogosort, bozosort,* kombinatorika, kompleksitas algoritma, permutasi.**

I. Introduction

Dalam ranah ilmu komputer—*computer science*—, salah satu algoritma yang paling terkenal dan sering digunakan oleh program adalah algoritma penyortiran, atau disebut juga sebagai algoritma pengurutan—*sorting algorithm*—. Algoritma penyortiran bertujuan untuk menyusun dan mengorganisasi data secara terstruktur, sehingga dapat mempermudah pengolahan dan analisis data.

Algoritma penyortiran yang sering digunakan adalah algoritma penyortiran yang bertujuan untuk mengurutkan angka dari yang terbesar ataupun yang terkecil, misalnya *bubble sort*, *heapsort*, *quicksort*, *merge sort*. Ada juga algoritma penyortiran yang mengurutkan kata berdasarkan abjad. Efisiensi sebuah algoritma penyortiran diukur berdasarkan kompleksitas waktu—*time complexity*— dan kompleksitas ruang memorinya—*space complexity*—. Kedua kompleksitas tersebut merupakan faktor dari sebuah kompleksitas algoritma. Masing-masing algoritma penyortiran mempunyai kompleksitas algoritma terburuk, kompleksitas algoritma rata-rata, dan kompleksitas algoritma terbaiknya sendiri.

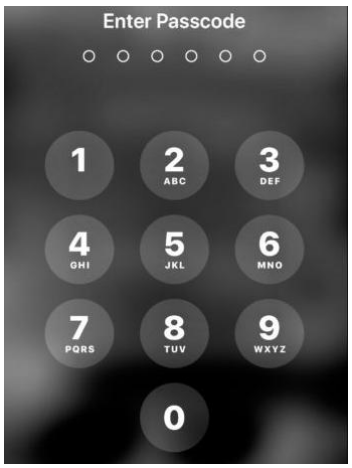
Ada beberapa algoritma penyortiran yang efisien dalam hal kompleksitas algoritma, karena waktu yang terpakai cukup singkat, dan tidak memakan banyak memori; ada juga beberapa algoritma yang kurang efisien karena memakan lebih banyak waktu dan/atau banyak memori. Namun, di samping itu, terdapat juga beberapa algoritma penyortiran tertentu yang memang secara sengaja dirancang untuk bekerja dengan tidak efisien sama sekali. Algoritma penyortiran seperti ini disebut dengan *esoteric sorting algorithm* (penyortiran algoritma esoteris). Salah satu algoritma penyortiran *esoteric* yang paling terkenal adalah *bogosort*, serta sebuah algoritma yang merupakan variasi dari *bogosort*, yakni *bozosort*.

Makalah ini dibuat dengan tujuan untuk menganalisis permutasi dan kompleksitas algoritma pada algoritma penyortiran *bogosort* dan *bozosort*.

II. Teori Dasar

A. Kombinatorika

Kombinatorial (*combinatorics*) adalah cabang yang mempelajari penyusunan objek. Lebih tepatnya, kombinatorial menghitung jumlah penyusunan objek tanpa mengenumerasi atau menghitung satu per satu secara manual semua kemungkinan susunan objek [1]. Contoh sederhananya adalah pin saat ingin membuka ponsel pintar—*smartphone*—; jika banyaknya digit pada pin adalah enam digit, maka ada berapa kemungkinan atau variasi enam buah angka yang dapat dimasukkan (anggap batas waktu saat salah memasukkan pin beberapa kali diabaikan atau tidak ada)?



Gambar 2.1. Pin enam digit pada ponsel pintar. Sumber: https://osxdaily.com/2019/08/18/how-disable-passcode-iphone-ipad/

1. Permutasi

Permutasi adalah banyaknya variasi-variasi pengaturan atau

penyusunan objek dengan melihat urutan objek tersebut. Permutasi dapat disimbolkan sebagai berikut:

- $P(n,r)$
- ${}_nP_r$
- ${}^nP_r$
- P_r^n

Permutasi dari r buah objek atau elemen yang berbeda, *dari n buah objek*, dapat dirumuskan sebagai

$$\boldsymbol{P(n,r) = \frac{n!}{(n-r)!}},$$

atau sebagai

$$P(n,r) = n \times (n-1) \times (n-2) \times \ldots \times (n-r+1)\,.$$

Jika seluruh elemen yang ada terpakai, yang berarti

$$r = n\,,$$

maka rumus tersebut dapat diturunkan menjadi

$$P(n,n) = \frac{n!}{(n-n)!}$$

$$P(n,n) = \frac{n!}{0!}$$

$$P(n,n) = \frac{n!}{1}\,;$$

atau dapat ditulis sebagai

$$P(n,n) = n \times (n-1) \times (n-2) \times \ldots \times (n-n+1)$$

$$P(n,n) = n \times (n-1) \times (n-2) \times \ldots \times 1\,,$$

sehingga permutasi dari n buah elemen berbeda dapat dirumuskan sebagai

$$\boldsymbol{P(n) = n!}\,.$$

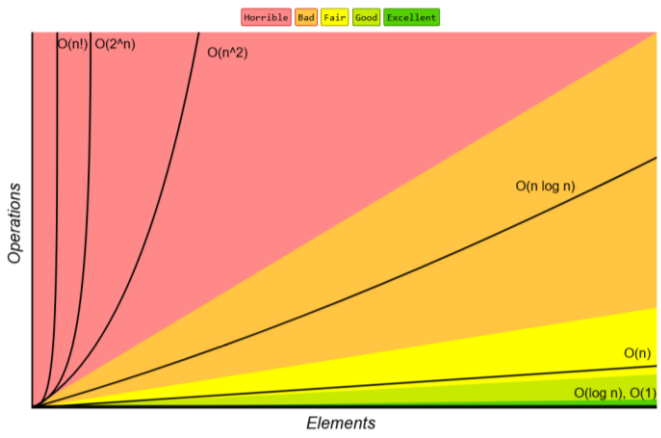
Jika terdapat beberapa elemen yang sama pada suatu kelompok atau himpunan elemen, maka permutasi dari n buah elemen dengan beberapa elemen yang sama dapat dirumuskan sebagai

$$\boldsymbol{P(n) = \frac{n!}{n_1! \times n_2! \times \ldots \times n_{k!}}},$$

dengan $n_1, n_2, \ldots, n_k$ adalah banyaknya tiap elemen (atau banyaknya tiap elemen yang banyaknya dua atau lebih) [2].

B. Kompleksitas Algoritma

Kompleksitas algoritma atau kompleksitas komputasional—*computational complexity*—adalah pengukuran seberapa efisien suatu program atau algoritma berlangsung sampai selesai. Sebuah algoritma yang baik tidak hanya mampu menyelesaikan suatu permasalahan, tetapi juga mampu menyelesaikannya secara sangkil atau efisien.

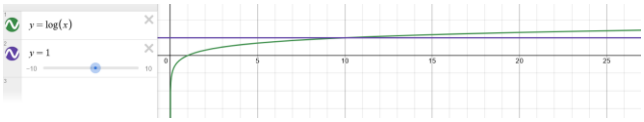


Gambar 2.2 Grafik kompleksitas algoritma. Sumber: https://www.bigocheatsheet.com

Keefisienan atau kesangkilan suatu program diukur dari waktu dan ruang memori yang terpakai selama algoritma tersebut berjalan. Pengukuran lama atau waktu sebuah algoritma berlangsung disebut sebagai kompleksitas waktu—*time complexity*—. Pengukuran banyak ruang memori yang dibutuhkan selama sebuah algoritma berlangsung disebut sebagai kompleksitas ruang—*space complexity*—[2].

Dapat dilihat pada grafik tersebut bahwa semakin menanjak sebuah garis, semakin tidak efisien algoritma tersebut. Algoritma yang paling tidak efisien adalah algoritma dengan kompleksitas $O(n!)$. Algoritma yang paling efisien adalah algoritma dengan kompleksitas $O(1)$.

Perhatikan bahwa $O(\log n)$ sebenarnya tidak sepenuhnya lurus atau selalu berimpit dengan $O(1)$. Jika kita perbesar, maka dapat dilihat pada gambar di bawah ini bahwa jumlah operasi $O(\log n)$ melewati jumlah operasi $O(1)$.



Gambar 2.3. Perbandingan grafik $O(1)$ dengan $O(\log n)$. Sumber: Dokumen Pribadi (https://www.desmos.com/calculator)

Perhatikan juga bahwa basis—*base*—dari logaritma yang sering digunakan pada $O(\log n)$ dan $O(n \times \log n)$ adalah basis 2. Namun, basis yang digunakan umumnya tidak berpengaruh, dapat menggunakan basis 2, basis 10, ataupun basis e (ln). Ini karena letak perbedaan nilai antara log dengan basis berbeda terdapat pada konstantanya. Sebagai contoh:

$$\log_2 n = \log_2 10 \times \log_{10} n$$

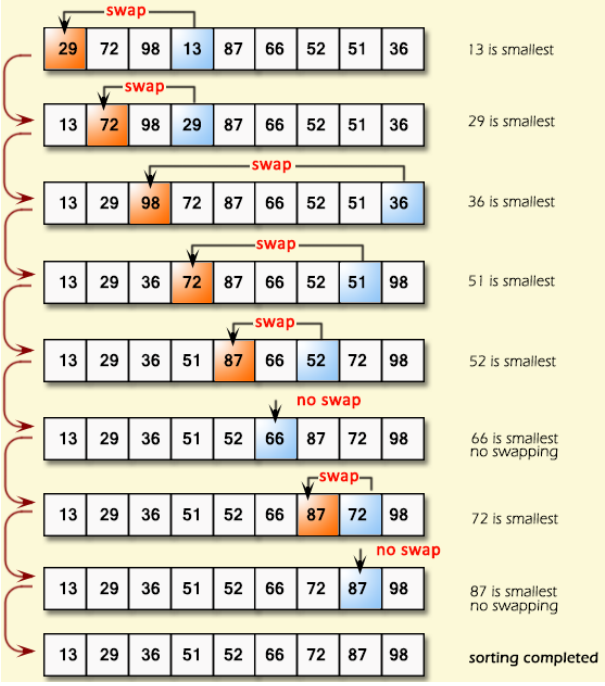
$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2}$$

$$\log_2 n \approx \frac{\log_{10} n}{0{,}301}\,,$$

dengan 0,301 sebagai konstanta. Pada notasi *Big O*, konstanta diabaikan, sehingga basis logaritma tidak berpengaruh [3].

C. Algoritma Penyortiran

Algoritma penyortiran atau algoritma pengurutan (*sorting algorithm*) adalah algoritma yang bertujuan untuk menyusun data secara berurut. Data ini umumnya merupakan bentuk angka. Namun, ada juga algoritma penyortiran huruf atau kata secara alfabet. Salah satu algoritma penyortiran yang termudah adalah *selection sort*.



Gambar 2.4. Algoritma penyortiran *selection sort*. Sumber: https://www.w3resource.com/python-exercises/data-structures-and-algorithms/python-search-and-sorting-exercise-5.php

Algoritma ini disebut mudah, karena kode implementasinya adalah sebagai berikut:

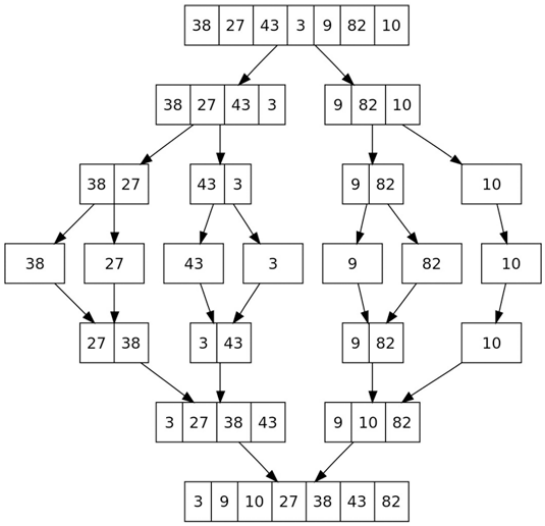
```
#Python - Bubble Sort

li = [2,9,3,1,6]
print(f"List belum tersortir: {li}")

for i in range (len(li)):
    for j in range (i+1, len(li)):
        if (li[j] < li[i]):
            li[i], li[j] = li[j], li[i]
print(f"List tersortir: {li}")
```

Namun, meskipun ini cukup mudah untuk diimplementasikan, algoritma ini kurang efisien, karena memiliki kompleksitas waktu $O(n^2)$, baik kompleksitas waktu terburuk, rata-rata, maupun terbaik. Kompleksitas waktu ini didapatkan dari iterasi linear (n) (`for j in range`) dalam sebuah iterasi linear (n) lainnya (`for i in range`), sehingga berupa n^2.

Algoritma penyortiran yang lebih efisien salah satunya adalah *merge sort*. Berikut ilustrasi algoritma penyortirannya:



Gambar 2.5. Algoritma penyortiran *merge sort*. Sumber: https://learncomputing.org/revision/gcse/2-1b

Algoritma ini lebih rumit daripada *selection sort*. Namun, algoritma penyortiran ini lebih sangkil. Kompleksitas waktu dari algoritma *merge sort*, baik kompleksitas waktu terburuk, rata-rata, maupun terbaik, adalah $O(n \times \log n)$. Bagian $\log n$ merujuk pada banyaknya tahapan pembagian *array* atau *list*. Misalnya pada Gambar 2.5., [38, 27, 43, 3, 9, 82, 10, x] dibagi dua menjadi [38, 27, 43, 3] dan [9, 82, 10, x]; ini merupakan pembagian tahap pertama. Setiap *array* yang ada dibagi dua lagi sampai menjadi setiap *array* yang hanya mempunyai satu anggota; terdapat tiga tahapan pembagian berdasarkan Gambar 2.5.. Ini berarti $\log n = \log_2 8 = 3$. Bagian n merujuk pada penggabungan *array*, karena setiap nilai atau bilangan dikunjungi secara linear (hanya dikunjungi sekali) [4].

1. Algoritma Penyortiran *Esoteric*

Menurut definisi Kamus Besar Bahasa Indonesia, esoterik atau *esoteric* mempunyai arti "bersifat khusus (rahasia, terbatas)" [5]. Ini berarti, algoritma penyortiran *esoteric* merupakan algoritma yang mengurutkan data secara khusus (dalam konteks ini, terbatas). Algoritma ini biasanya dibuat sebagai lelucon atau candaan.

Ada beberapa jenis algoritma penyortiran yang bersifat *esoteric*, contohnya:

- *Bogosort*
  Bogosort adalah algoritma yang menyortir data (berupa angka) dengan cara mengacak-acak data tersebut sampai semua data terurut [6].
- *Bozosort*
  Bozosort adalah algoritma yang mengurutkan data dengan cara memilih dua buah data (berupa angka) secara acak, dan menukar posisi kedua data tersebut. Algoritma ini melakukan hal tersebut sampai semua data terurut [6].
- *Miracle sort*
  Miracle sort adalah algoritma "penyortiran" yang mana penggunanya harus menunggu dan berharap agar sebuah pengaruh luar atau sebuah keajaiban memodifikasi bit pada komputer sehingga data menjadi terurut. Dalam

kata lain, algoritma ini hanya mengecek apakah data terurut, dan tidak melakukan penyortiran sama sekali [6]. Dapat dikatakan bahwa kompleksitas waktu algoritma ini adalah $O(\infty)$ jika data tidak terurut.

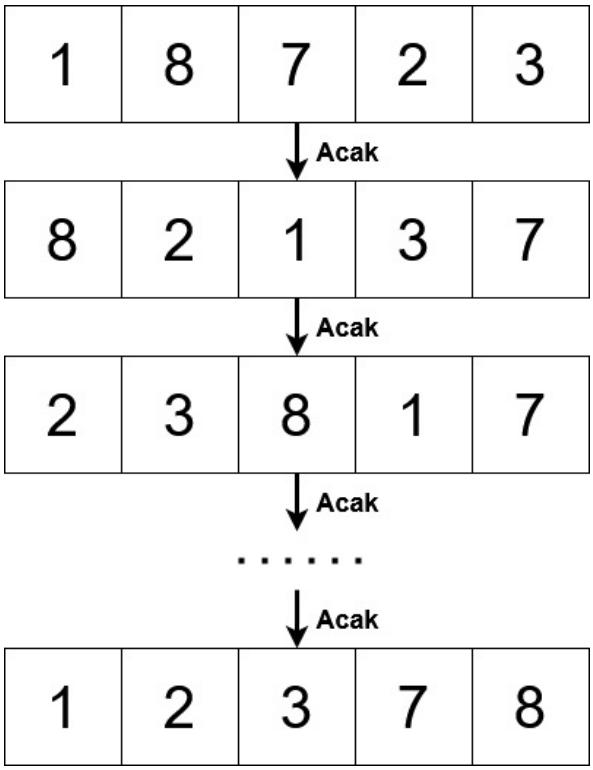
- *Bogobogosort*
  Bugologist adalah algoritma yang jauh lebih tidak efisien daripada *bogosort*. Cara kerjanya adalah melakukan *bogosort* satu elemen pertama, lalu *bogosort* dua elemen pertama, sampai seterusnya hingga semua elemen dilakukan *bogosort*. Jika selama proses tersebut, data tidak terurut, maka akan diulang dari awal lagi (dimulai dari *bogosort* satu elemen pertama lagi) [6].

Dari beberapa algoritma penyortiran *esoteric* yang ada, algoritma yang akan dianalisis adalah algoritma penyortiran *bogosort* dan *bozosort*.

III. Analisis Algoritma Penyortiran *Bogosort* dan *Bozosort*

A. Cara Kerja Bogosort

Bogosort adalah sebuah algoritma penyortiran *esoteric* yang mengacak-acak data berupa angka sampai data tersebut terurut sepenuhnya. Cara kerjanya dapat diilustrasikan sebagai berikut:



Gambar 3.1. Ilustrasi cara kerja *bogosort*. Sumber: Dokumen Pribadi

Untuk mengimplementasikan kode *bogosort*, diperlukan *module* `random`. *Module* ini digunakan karena berdasarkan cara kerja *bogosort*, data disusun-susun ulang secara acak (*random*).

```
# Python - Bogosort

import random
from time import perf_counter

def isSorted(li):
    n = len(li)
    for i in range(0, n-1):
        if (li[i] > li[i+1]):
            return False
    return True

def bogoSort(li):
    n = len(li)
    count = 0
    start = perf_counter()
    while not isSorted(li):
        for i in range(0, n):
            j = random.randint(0, n-1)
            li[i], li[j] = li[j], li[i]
        count += 1
    end = perf_counter()
    print(f"Jumlah pengacakan total:
{count}")
    print(f"Waktu yang diperlukan: {end-
start}")

# input
while True:
    li = input("Masukkan bilangan, dipisah
dengan koma: ").replace(" ", "").split(",")
    valid = True
    for i, num in enumerate(li):
        if num.isdigit():
            li[i] = int(num)
        else:
            valid = False
            break
    if valid:
        break
print(f"Sebelum diurutkan: {li}")

# Mulai proses bogosort
if valid:
   bogoSort(li)
   print(f"Setelah diurutkan: {li}")
```

Menggunakan kode tersebut, jika *array* yang diinput adalah [3, 5, 4, 1, 2, 5, 6, 9, 7, 12], maka hasilnya adalah seperti ini:

```
Masukkan bilangan, dipisah dengan koma: 3,5,4,1,2,5,6,9,7,12
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12]
Jumlah pengacakan total: 646317
Waktu yang diperlukan: 3.109847699990496
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12]
```

Gambar 3.2. Hasil *test case bogosort* dengan sepuluh buah data. Sumber: Dokumen Pribadi

Jika *array* memiliki sepuluh buah data, waktu yang diperlukan adalah sekitar 3,1 detik. Namun, jika menambahkan satu data ke dalam *array* tersebut sehingga menjadi sebelas data, maka waktu dan jumlah pengacakan dapat menjadi jauh lebih

besar. Misalnya, bilangan 14 ditambahkan ke *array* sebelumnya, sehingga menjadi [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14], maka hasilnya adalah sebagai berikut:

```
Masukkan bilangan, dipisah dengan koma: 3,5,4,1,2,5,6,9,7,12,14
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14]
Jumlah pengacakan total: 10106341
Waktu yang diperlukan: 53.10888169996906
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12, 14]
```

Gambar 3.3. Hasil *test case bogosort* dengan sebelas buah data. Sumber: Dokumen Pribadi

Perhatikan hasil *test case* dengan sebelas data tersebut. Waktu penyortiran sebelas buah data tersebut adalah sekitar 17 kali waktu penyortiran sepuluh buah data, yaitu 53,1 detik. Itu hanya salah satu percobaan dengan sebelas buah data. Jika beruntung, maka data dapat tersortir dengan lebih cepat. Jika tidak, maka data dapat tersortir dengan jauh lebih lama.

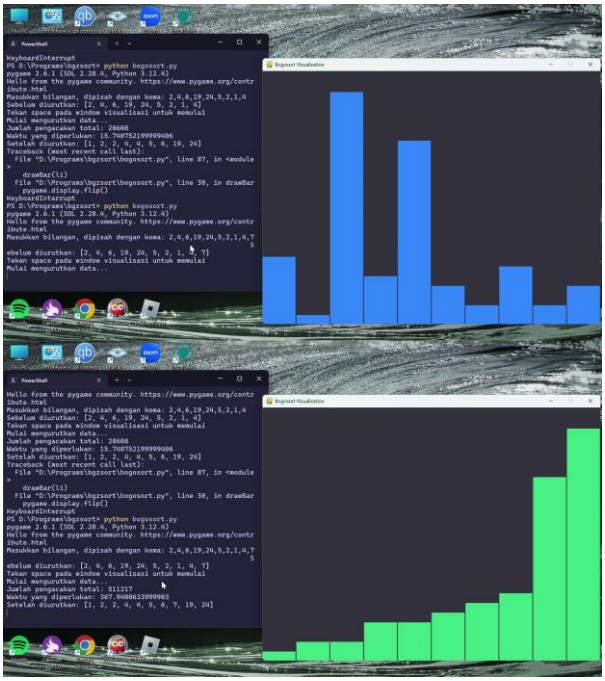
```
Masukkan bilangan, dipisah dengan koma: 3,5,4,1,2,5,6,9,7,12,14
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14]
Jumlah pengacakan total: 2544649
Waktu yang diperlukan: 16.774720599998545
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12, 14]
```

Gambar 3.3. Hasil *test case* kedua *bogosort* dengan sebelas buah data yang sama. Sumber: Dokumen Pribadi

```
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14]
Jumlah pengacakan total: 41024508
Waktu yang diperlukan: 262.5809611000004
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12, 14]
```

Gambar 3.4. Hasil *test case* ketiga *bogosort* dengan sebelas buah data yang sama. Sumber: Dokumen Pribadi

Berikut adalah percobaan dengan sepuluh buah data lain, dengan visualisasi:



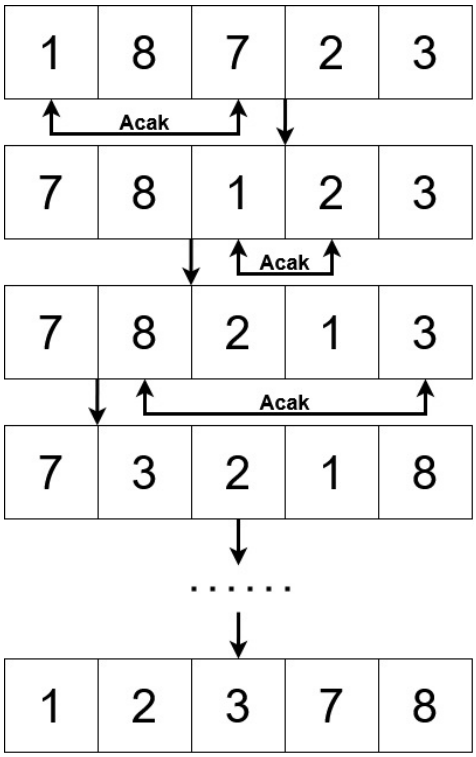
Gambar 3.5. Hasil *test case bogosort* dengan visual. Sumber: Dokumen Pribadi (video YouTube: https://youtu.be/F1rWZYvE5yU)

Dapat dilihat bahwa waktu yang dibutuhkan adalah 367,94 detik untuk menyortir sepuluh buah data tersebut.

Waktu penyortiran untuk data yang lebih banyak dapat memakan waktu yang jauh lebih besar, sehingga algoritma penyortiran ini disebut sangat tidak efisien.

B. Cara Kerja Bozosort

Bozosort adalah sebuah algoritma penyortiran *esoteric* yang memilih dua buah data (angka) secara acak dan menukar posisi kedua data tersebut. Proses ini berulang sampai data terurut. Cara kerjanya adalah sebagai berikut:



Gambar 3.4. Ilustrasi cara kerja *bozosort*. Sumber: Dokumen Pribadi

Pengimplementasian algoritma penyortiran *bozosort* adalah sebagai berikut:

```
# Python - Bozosort

import random
from time import perf_counter

def isSorted(li):
    n = len(li)
    for i in range(0, n-1):
        if (li[i] > li[i+1]):
            return False
    return True

def bozoSort(li):
    n = len(li)
    count = 0
```

```
    start = perf_counter()
    while not isSorted(li):
        i = random.randint(0, n-1)
        j = random.randint(0, n-1)
        li[i], li[j] = li[j], li[i]
        count += 1
    end = perf_counter()
    print(f"Jumlah pertukaran posisi total: {count}")
    print(f"Waktu yang diperlukan: {end-start}")

# input
while True:
    li = input("Masukkan bilangan, dipisah dengan koma: ").replace(" ", "").split(",")
    valid = True
    for i, num in enumerate(li):
        if num.isdigit():
            li[i] = int(num)
        else:
            valid = False
            break
    if valid:
        break
print(f"Sebelum diurutkan: {li}")

# Mulai proses bozosort
bozoSort(li)
print(f"Setelah diurutkan: {li}")
```

Menggunakan kode tersebut, jika *array* yang diinput adalah data dari percobaan *bogosort*, yaitu [3, 5, 4, 1, 2, 5, 6, 9, 7, 12], maka hasilnya adalah seperti ini:

```
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12]
Jumlah pertukaran posisi total: 5686941
Waktu yang diperlukan: 8.105530800006818
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12]
```

Gambar 3.5. Hasil *test case bozosort* dengan sepuluh buah data. Sumber: Dokumen Pribadi

Jika *array* memiliki sepuluh buah data, waktu yang diperlukan adalah sekitar 8,1 detik. Namun, jika menambahkan satu data ke dalam *array* tersebut sehingga menjadi sebelas data, maka waktu dan jumlah pengacakan dapat menjadi jauh lebih besar. Misalnya, menggunakan data kedua dari *bogosort*, maka hasilnya adalah sebagai berikut:

```
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14]
Jumlah pertukaran posisi total: 39710159
Waktu yang diperlukan: 82.12329019999743
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12, 14]
```

Gambar 3.6. Hasil *test case bozosort* dengan sebelas buah data. Sumber: Dokumen Pribadi

Perhatikan hasil *test case* dengan sebelas data tersebut. Waktu penyortiran sebelas buah data adalah sekitar 10 kali waktu penyortiran sepuluh buah data, yaitu 82,1 detik. Itu hanya salah satu percobaan dengan sebelas buah data. Jika beruntung, maka data dapat tersortir dengan lebih cepat. Jika tidak, maka data dapat tersortir dengan jauh lebih lama.

```
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14]
Jumlah pertukaran posisi total: 2695716
Waktu yang diperlukan: 3.7606655999989016
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12, 14]
```

Gambar 3.7. Hasil *test case* kedua *bozosort* dengan sebelas buah data yang sama. Sumber: Dokumen Pribadi

```
Sebelum diurutkan: [3, 5, 4, 1, 2, 5, 6, 9, 7, 12, 14]
Jumlah pertukaran posisi total: 44942472
Waktu yang diperlukan: 130.21520050000254
Setelah diurutkan: [1, 2, 3, 4, 5, 5, 6, 7, 9, 12, 14]
```

Gambar 3.8. Hasil *test case* ketiga *bozosort* dengan sebelas buah data yang sama. Sumber: Dokumen Pribadi

Sama seperti *bogosort*, waktu penyortiran untuk data yang lebih banyak dapat memakan waktu yang jauh lebih besar, sehingga algoritma penyortiran *bozosort* juga sangat tidak efisien.

C. Analisis Permutasi dari Bogosort *dan* Bozosort

Sebanyak $r = n$ buah data dari n data digunakan pada *bogosort*. Dalam kata lain, semua data terpakai. Ini berarti, permutasi dari *bogosort* dan *bozosort* adalah

$$P(n) = n!$$

jika datanya berbeda semua.

Pada data pertama, yaitu sebanyak sepuluh buah data yang berbeda semua, banyaknya variasi data yang ada adalah

$$P(10) = 10!$$
$$P(10) = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$$
$$P(10) = 3.628.800\,.$$

Ini berarti ada 3.628.800 variasi *array* atau urutan untuk sepuluh data berbeda. Hal ini berarti ada 1 dari 3.628.800 variasi urutan yang benar, atau sekitar $2{,}75 \times 10^{-7}\%$ (lebih tepatnya 0,0000002755732%) kemungkinan benar.

Pada data kedua, yaitu sebanyak sebelas buah data dengan satu bilangan yang banyaknya adalah dua, banyaknya variasi data yang ada adalah

$$P(11) = \frac{11!}{2!}$$
$$P(11) = 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3$$
$$P(11) = 19.958.400\,.$$

Hal ini berarti ada 19.958.400 variasi *array* atau urutan untuk sebelas data tersebut. Hal ini berarti ada 1 dari 19.958.400 variasi urutan yang benar, atau sekitar $5{,}01 \times 10^{-8}\%$ (lebih tepatnya 0,0000000501042%) kemungkinan benar.

D. Analisis Kompleksitas Algoritma dari Bogosort *dan* Bozosort

Pada algoritma *bogosort* dan *bozosort*, karena kemungkinan data terurutnya sangat kecil, maka algoritma ini sangat tidak efisien. Untuk membuktikannya, akan dilakukan analisis kompleksitas algoritma dari kedua algoritma tersebut.

Pada algoritma penyortiran *bogosort*, jika semua data sudah terurut, maka kompleksitas waktu pengacakannya adalah $O(1)$, karena tidak dilakukan pengacakan.

Dalam kasus rata-rata, kompleksitas waktu pengacakannya adalah $O(n \times n!)$. Perhatikan cuplikan kode berikut:

```
# ...
for i in range(0, n):
    j = random.randint(0, n-1)
    li[i], li[j] = li[j], li[i]
# ...
```

$n!$ berasal dari rumus permutasi n buah objek (dalam kasus ini, data atau angka), yaitu

$$P(n) = n!,$$

sedangkan n berasal dari *loop* linear (`for i in range(0, n)`).

Untuk kasus terburuknya, *bogosort* akan berlangsung sangat lama. *Bogosort* bisa saja mengacak data sampai bertahun-tahun, terutama jika data yang ingin disortir sangat banyak atau luas. Kompleksitas waktu untuk kasus terburuk sulit ditentukan dan dapat dianggap sebagai $O(\infty)$. Sebagai contoh, dapat dikatakan, "sudah ratusan tahun berlalu, namun *bogosort* ini masih belum selesai mengurutkan angka yang sangat banyak ini. Harus sampai kapan algoritma ini berhasil mengurutkan angka-angkanya?"

Kompleksitas waktu terbaik dan terburuk dari algoritma penyortiran *bozosort* adalah sama dengan *bogosort*. Pada algoritma penyortiran *bozosort*, kompleksitas waktu terbaik pertukaran datanya adalah $O(1)$, yaitu jika data sudah terurut.

Sedangkan kasus terburuk pertukaran datanya adalah $O(\infty)$ juga, karena dapat berlangsung sangat lama dan sulit untuk menentukannya, seperti *bogosort*.

Untuk kasus rata-rata pertukaran datanya, kompleksitas algoritmanya adalah $O(n!)$. Perhatikan cuplikan kode berikut:

```
# ...
    i = random.randint(0, n-1)
    j = random.randint(0, n-1)
    li[i], li[j] = li[j], li[i]
# ...
```

Faktorial $n!$ berasal dari permutasi r buah data atau angka dari n buah data, yaitu

$$P(n,r) = \frac{n!}{(n-r)!}.$$

Dalam kasus ini, $r = 2$ karena diambil dua buah bilangan, sehingga

$$P(n,2) = \frac{n!}{(n-2)!}.$$

Dalam notasi Big O, ini ditulis sebagai $O(n!)$.

Kompleksitas ruang dari algoritma penyortiran *bogosort* dan *bozosort* adalah sama, yaitu $O(1)$. Ini karena *bogosort* dan *bozosort* menggunakan dan mengolah *array* atau kumpulan data yang sama dan menyimpan hasilnya pada *array* tersebut secara langsung. Kedua algoritma ini tidak membuat data atau *array* yang baru.

IV. Kesimpulan

Algoritma penyortiran bogosort dan bozosort merupakan algoritma yang mengurutkan data secara tidak efisien. *Bogosort* mengacak-acak semua data sampai semua datanya terurut. *Bozosort* memilihi dua data secara acak dan menukar posisi keduanya.

V. Lampiran

Berikut adalah video demonstrasi dari *bogosort* beserta dengan visual: https://youtu.be/F1rWZYvE5yU.

File kode dari *bogosort* dan *bozosort*, beserta dengan visualisasinya: https://drive.google.com/drive/folders/1JSoG__Is7Yc3-Ro9bLt7kWIeBoJpREte

Visualisasi algoritma penyortiran *bogosort* dan *bozosort* menggunakan *library* PyGame: https://github.com/pygame/pygame

VI. Ucapan Terima Kasih

Penulis ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa, orang tua penulis, serta Bapak dan Ibu dosen mata kuliah Matematika Diskrit Institut Teknologi Bandung atas dukungan dan rahmat kepada penulis, sehingga penulis dapat menulis makalah ini dan menyelesaikannya.

Referensi

[1] R. Munir, "Kombinatorika Bagian 1," Homepage Rinaldi Munir, 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian1-2024.pdf (accessed Jan. 04, 2025).

[2] K. H. Rosen, *Discrete Mathematics and Its Applications*, 8th ed. New York, Ny: Mcgraw-Hill, 2019, pp. 428–431, 231.

[3] BuckFilledPlatypus, "Is Big O(logn) log base e?," *Stack Overflow*, Oct. 15, 2009. https://stackoverflow.com/a/1569710 (accessed Jan. 05, 2025).

[4] Supercat, "Why is merge sort worst case run time O (n log n)?," *Stack Overflow*, Oct. 18, 2011. https://stackoverflow.com/a/7801883 (accessed Jan. 05, 2025).

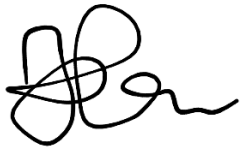
[5] "Esoterik - KBBI VI Daring," *KBBI VI Daring*, Oct. 28, 2023. https://kbbi.kemdikbud.go.id/entri/esoterik (accessed Jan. 06, 2025).

[6] V. Dai, "Esoteric Sorting Algorithms," *Vivian Dai's Blog*, Apr. 15, 2022. https://viviandai.hashnode.dev/esoteric-sorting-algorithms (accessed Jan. 06, 2025).

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 Desember 2024



Jonathan Levi, 13523132