Tugas Besar

Analisis Efisiensi Algoritma Quick Sort Iteratif dan Rekursif Pada Pengurutan Skor Game dalam Leaderboard

Dibuat oleh :

Abdurrohman Robbany – 1304211004

PROGRAM STUDI S1 INFORMATIKA PJJ

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM

BANDUNG

2024

Studi Kasus Permasalahan

Sebuah game pasti memiliki system leaderboard untuk menampilkan peringkat pemain berdasarkan skor yang dicapai. Leaderboard adalah fitur penting dalam game yang memberikan motivasi tambahan kepada pemain untuk mencapai skor tertinggi dan bersaing dengan pemain lain. Dalam Tugas Besar ini, fokus akan diberikan pada cara efisiensi untuk mengurutkan skor pemain dalam leaderboard agar dapat diakses dan ditampilkan dengan cepat.

Algoritma yang dipilih

Quick Sort adalah algoritma pengurutan yang menggunakan strategi bagi-dan-kuasai untuk membagi dan mengurutkan array. Quick Sort bekerja seperti membagi pekerjaan besar menjadi tugas-tugas kecil yang lebih mudah. Dalam leaderboard game, di mana skor pemain selalu berubah, Quick Sort membantu mengurutkan skor dengan cepat saat daftar skor diperbaharui. Algoritma ini tidak sulit untuk dipahami dan diimplementasikan sehingga memudahkan pengembang game dalam menerapkannya. Versi pendekatan iteratif dan rekursif dari Quick Sort dapat disesuaikan dengan baik dengan kebutuhan pengurutan skor game dalam Leaderboard.

1. Iterative quick sort
   Data skor player akan dibagi berdasarkan elemen pivot secara berulang menggunakan pendekatan iterative. Setiap iterasi akan membagi data menjadi dua bagian, yaitu yang lebih kecil dari pivot dan yang lebih besar dari pivot sehingga membentuk ukuran skor yang bener secara bertahap.

2. Rekursif quick sort
   Dalam pendekatan rekursif, proses pengurutan skor dilakukan dengan cara memecah masalah menjadi submasalah yang lebih kecil hingga mencapai kondisi dasar(base case). Pendekatan rekursif akan membagi data skor player berdasarkan elemen pivot secara rekursif, di mana setiap pemanggilan rekursif akan membagi data menjadi dua bagian yang lebih kecil sampai seluruh data terurut.

Implementasi

Untuk mengetahui running time tercepat dan efisien dari kedua pendekatan pada algoritma quick sort, maka perlu dilakukan uji coba menggunakan bahasa pemograman python atau C++ agar memudahkan penguijiannya. Dalam hal ini saya menggunakan bahasa pemograman python. Pada program pertama, di isikan skor player yang random dengan kapasitas inputan datanya 10.000 . Pada program yang kedua, di isikan list player dan skornya. Nantinya kedua program tersebut memiliki outputan yang dapat mengetahui berapa jumlah running time yang dihasilkan.

Pada program pertama, diberikan fungsi quick_sort iteratif dan rekursif yang dapat menghitung running time dalam mengurutkan skor yang random dengan range inputan datanya 100, 500, 1000, 5000, dan 10000 .

```python
# Fungsi Partition untuk pembagian elemen pivot
def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

# Fungsi Quick Sort Iteratif
def quick_sort_iterative(arr):
    stack = []
    stack.append((0, len(arr) - 1))

    while stack:
        low, high = stack.pop()
        if low < high:
            pivot = partition(arr, low, high)
            stack.append((low, pivot - 1))
            stack.append((pivot + 1, high))
```

```python
# Fungsi Quick Sort Rekursif
def quick_sort_recursive(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        less = [x for x in arr[1:] if x <= pivot]
        greater = [x for x in arr[1:] if x > pivot]
        return quick_sort_recursive(less) + [pivot] + quick_sort_recursive(greater)
```

```python
# Fungsi untuk membuat data skor pemain secara acak
def generate_random_scores(n):
    return [random.randint(1, 1000) for _ in range(n)]

# Fungsi untuk mengukur waktu eksekusi
def measure_execution_time(sort_func, arr):
    start_time = time.time()
    sort_func(arr)
    return time.time() - start_time

# Ukuran data skor pemain yang akan diuji
input_sizes = [100, 500, 1000, 5000, 10000]
iterative_runtimes = []
recursive_runtimes = []

for size in input_sizes:
    scores = generate_random_scores(size)

    iterative_time = measure_execution_time(quick_sort_iterative, scores.copy())
    recursive_time = measure_execution_time(quick_sort_recursive, scores.copy())

    iterative_runtimes.append(iterative_time)
    recursive_runtimes.append(recursive_time)


#running time yang didapat
for i in range(len(input_sizes)):
    print(f"Running time iteratif untuk {input_sizes[i]} data: {iterative_runtimes[i]:.4f} detik")
    print(f"Running time rekursif untuk {input_sizes[i]} data: {recursive_runtimes[i]:.4f} detik")
```

Pada program kedua, diberikan fungsi quick_sort iteratif dan rekursif yang dapat menghitung running time dari pengurutan skor yang telah diberi.

```python
# Algoritma Quick Sort Iteratif
def quicksort_iterative(arr):
    if len(arr) <= 1:
        return arr

    stack = []
    stack.append((0, len(arr) - 1))

    while stack:
        low, high = stack.pop()
        if low < high:
            pivot = arr[high][1]
            i = low - 1
            for j in range(low, high):
                if arr[j][1] < pivot:
                    i += 1
                    arr[i], arr[j] = arr[j], arr[i]
            arr[i + 1], arr[high] = arr[high], arr[i + 1]
            stack.append((low, i))
            stack.append((i + 2, high))

    return arr
```

```python
# Algoritma Quick Sort Rekursif
def quicksort_recursive(arr):
    if len(arr) <= 1:
        return arr

    pivot = arr[random.randint(0, len(arr) - 1)][1]
    lesser = [x for x in arr if x[1] < pivot]
    equal = [x for x in arr if x[1] == pivot]
    greater = [x for x in arr if x[1] > pivot]

    return quicksort_recursive(lesser) + equal + quicksort_recursive(greater)
```

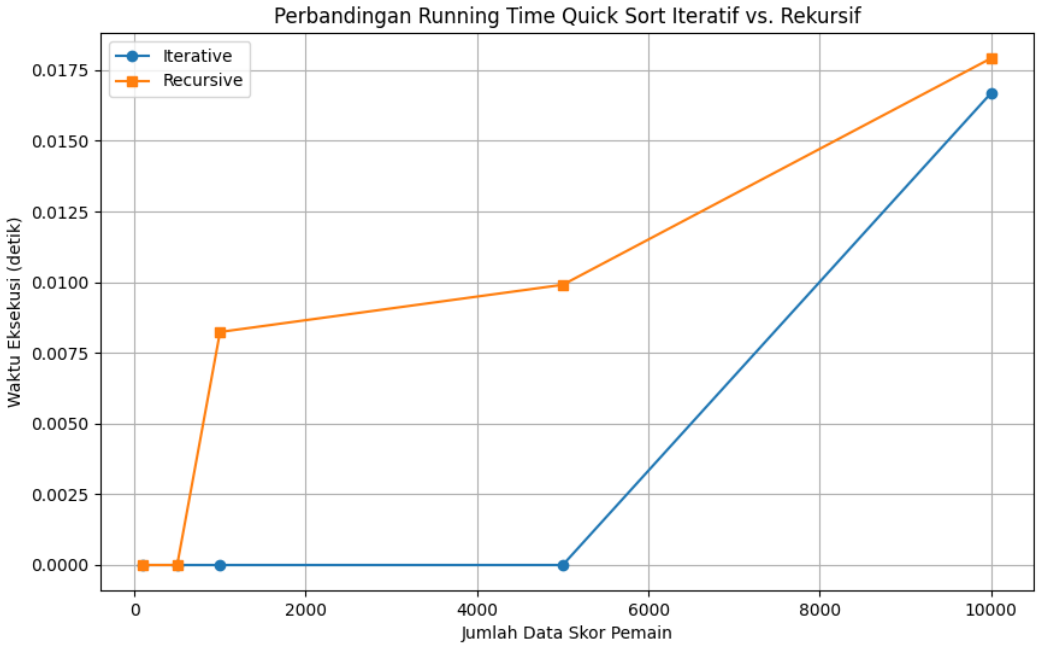
```python
# List pemain dan skor
players = [("byne", 450), ("kaniee", 680), ("stinger", 110), ("yui", 300), ("bibo", 888),
           ("kaniee", 1150), ("baubau", 770), ("miko", 798), ("blast", 500), ("baka", 90),
           ("junkie", 320), ("demonix", 662), ("leon", 760), ("drago", 1052), ("phoenic", 1102),
           ("chama", 1055), ("fubuchan", 999)]

# Pengurutan skor dengan algoritma Quick Sort Iteratif
sorted_scores_iterative = quicksort_iterative(players.copy())
iterative_time = timeit.timeit(lambda: quicksort_iterative(players.copy()), number=1000)

# Pengurutan skor dengan algoritma Quick Sort Rekursif
sorted_scores_recursive = quicksort_recursive(players.copy())
recursive_time = timeit.timeit(lambda: quicksort_recursive(players.copy()), number=1000)

# Plotting running time
labels = ['Iterative', 'Recursive']
times = [iterative_time, recursive_time]
```

Pertama-tama skor player disimpan dalam bentuk tupel, lalu dilakukan pengurutan menggunakan kedua pendekatan. Setelah itu waktu eksekusi masing-masing pendekatan akan diukur dan hasilnya akan ditampilkan dalam grafik perbandingan waktu eksekusi.

Grafik Perbandingan Running Time

Grafik yang dihasilkan dari pengujian diatas adalah sebagai berikut.

1. Pada program pertama:

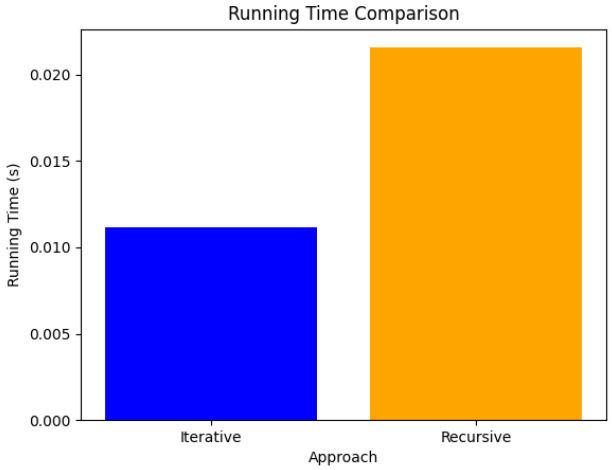


Dan running timenya

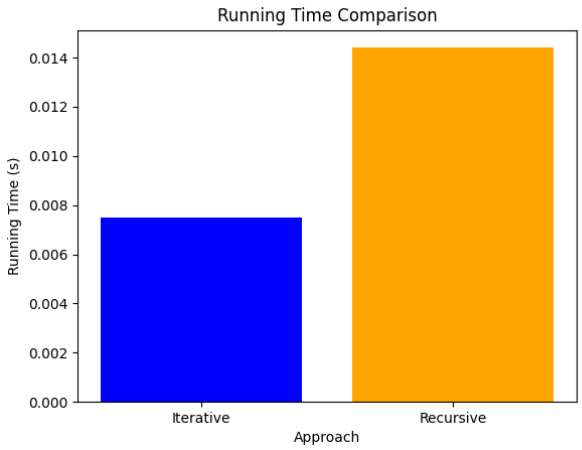
```
Running time iteratif untuk 100 data: 0.0000 detik
Running time rekursif untuk 100 data: 0.0000 detik
Running time iteratif untuk 500 data: 0.0000 detik
Running time rekursif untuk 500 data: 0.0000 detik
Running time iteratif untuk 1000 data: 0.0000 detik
Running time rekursif untuk 1000 data: 0.0082 detik
Running time iteratif untuk 5000 data: 0.0000 detik
Running time rekursif untuk 5000 data: 0.0099 detik
Running time iteratif untuk 10000 data: 0.0167 detik
Running time rekursif untuk 10000 data: 0.0179 detik
```

Pada program pertama ini hasilkan akan berbeda-beda dalam setiap run nya karena angka input yang diberikan adalah random.

2. Pada program kedua:




```
Hasil pengurutan skor:
Iterative: [('baka', 90), ('stinger', 110), ('yui', 300), ('junkie', 320), ('byne', 450), ('blast', 500), ('demonix', 662), ('kaniee', 680), ('leon', 760), ('baubau', 770), ('miko', 798
), ('bibo', 888), ('fubuchan', 999), ('drago', 1052), ('chama', 1055), ('phoenic', 1102), ('kaniee', 1150)]
Recursive: [('baka', 90), ('stinger', 110), ('yui', 300), ('junkie', 320), ('byne', 450), ('blast', 500), ('demonix', 662), ('kaniee', 680), ('leon', 760), ('baubau', 770), ('miko', 798
), ('bibo', 888), ('fubuchan', 999), ('drago', 1052), ('chama', 1055), ('phoenic', 1102), ('kaniee', 1150)]
Running Time Iteratif:  0.011184999952092767
Running Time Rekursif:  0.0215586000121299
```




```
Hasil pengurutan skor:
Iterative: [('baka', 90), ('stinger', 110), ('yui', 300), ('junkie', 320), ('byne', 450), ('blast', 500), ('demonix', 662), ('kaniee', 680), ('leon', 760), ('baubau', 770), ('miko', 798
), ('bibo', 888), ('fubuchan', 999), ('drago', 1052), ('chama', 1055), ('phoenic', 1102), ('kaniee', 1150)]
Recursive: [('baka', 90), ('stinger', 110), ('yui', 300), ('junkie', 320), ('byne', 450), ('blast', 500), ('demonix', 662), ('kaniee', 680), ('leon', 760), ('baubau', 770), ('miko', 798
), ('bibo', 888), ('fubuchan', 999), ('drago', 1052), ('chama', 1055), ('phoenic', 1102), ('kaniee', 1150)]
PS C:\Users\USER>
```

Setelah menjalankan program, kita dapat melihat output berupa running time untuk kedua pendekatan beserta grafik perbandingan waktu eksekusi antara pendekatan iteratif dan rekursif sebagaimana ditunjukkan pada gambar diatas. Dengan melihat running time yang telah diukur, kita dapat menentukan pendekatan mana yang lebih efektif untuk studi kasus pengurutan skor game dalam leaderboard ini.

Analisis Perbandingan Kedua Algoritma

Pada kasus algoritma QuickSort, waktu eksekusi algoritma secara iteratif dan rekursif biasanya memiliki nilai kompleksitas waktu rata-rata O(n log n) dalam kasus terbaik. Namun, dalam kasus terburuk algoritma ini dapat memiliki kompleksitas waktu O(n^2). Dari hasil kedua program yang telah berikan, terlihat bahwa pendekatan iteratif memiliki running time yang lebih cepat dibandingkan dengan pendekatan rekursif untuk semua ukuran data skor player yang diuji.

Efisiensi waktu yang diperoleh pada pendekatan iteratif terhadap pendekatan rekursif pada algortima quick sort dapat dihitung dan dianalisis berdasarkan perbandingan running time untuk setiap ukuran data yang diuji. Efisiensi waktu iteratif terhadap rekursif dapat berbeda untuk setiap ukuran data, namun secara umum pendekatan iteratif terbukti lebih efisien dalam hal waktu eksekusi. Meskipun hasil running time yang diperoleh tidak selalu mencerminkan kompleksitas waktu secara teoritis, kompleksitas waktu dari algoritma QuickSort baik iteratif maupun rekursif, yang paling mendekati berdasarkan hasil running time tersebut adalah O(n log n). Hal ini menunjukkan bahwa algoritma Quick sort secara efisien dapat mengurutkan data bahkan untuk data(ukuran data) dalam skala besar.

Kesimpulan

Berdasarkan hasil running time yang diperoleh, pendekatan iterative pada algortima quicksort menunjukkan performa lebih baik daripada pendekatan rekursif dalam hal waktu eksekusi. Algoritma quicksort dengan pendekatan iteratif terbukti menawarkan solusi yang lebih efisien dan cepat dalam pengurutan data dibandingkan dengan pendekatan rekursif, terutama pada dataset yang lebih besar. Pendekatan iteratif memiliki performa yang lebih unggul.

REFERENSI

https://cs.stackexchange.com/questions/3/why-is-quicksort-better-than-other-sorting-algorithms-in-practice

https://www.geeksforgeeks.org/iterative-quick-sort/

https://www.geeksforgeeks.org/python-program-for-iterative-quick-sort/

https://www.geeksforgeeks.org/python-program-for-quicksort/

https://builtin.com/articles/quicksort

https://medium.com/@elizabeth.michelee/cara-membuat-grafik-dengan-matplotlib-dalam-python-ef41bdf2695d

https://stackoverflow.com/questions/75076584/quicksort-iterative-animation

https://www.studytonight.com/python-programs/python-program-for-iterative-quicksort

https://stackoverflow.com/questions/20175380/quick-sort-python-recursion