Complexity

It’s complex but yet not too complex

Warning

Slide ini bukanlah slide yang mengacu kepada kisi-kisi bahan ujian, melainkan untuk lebih menjelaskan mengenai topik tersebut.

Baca kalau mau baca ☺

Prerequisite

Bisa itung-itungan anak tk, sd, smp

Udah mandi & makan

Bisa baca koding

Mau baca koding

Apa itu kompleksitas?

You can read the definition in here:

https://en.wikipedia.org/wiki/Analysis_of_algorithms

Nah, pastinya gaada yang baca :v

Intinya menghitung secara kasar berapa ***proses*** yang diperlukan untuk menjalankan suatu program.

Biasanya proses yang akan dihitung kompleksitasnya ada 2 jenis:

1. Memori
2. Waktu

Kompleksitas Memori

Kompleksitas memori membahas tentang perkiraan berapa memori yang dipakai dalam program tersebut.

Kenapa perlu hitung?

Kasus: Anggep aja, windows ngadain update nih. Jadi buat jalanin windows aja, perlu memori 8 GB. Gara-gara itu, kalian uda gabisa jalanin apapun lagi (Buka notepad aja gabisa, gausa mimpi maen dota 2).

So it’s important ☺

Menghitung Kompleksitas Memori

Big Oh, Theta, Omega? Belum waktunya cuy

Tinggal jumlahin aja berapa besar memori tipe data dari setiap variabel yang dipakai

Menghitung Kompleksitas Memori

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)

int A = 5 //Karena integer, maka butuh 4 byte
float x[4] //Karena float dan array size 4, maka butuh 4 * 4 = 16 byte

Kompleksitas Waktu

Kompleksitas waktu membahas tentang berapa lama perkiraan program tersebut berjalan.

Kenapa perlu kompleksitas waktu?

Kasus: Kita semua tahu, google tuh datanya banyak. Anggep aja kalian lagi cari kata “kompleksitas” di google. Kalau google mencari “kompleksitas” secara satu persatu, maka dalam 1 tahun pun tidak akan selesai.

So it’s important ☺

Dengan tahu kompleksitas waktu dan memori, kita bisa mengetahui bahkan sebelum program tersebut dibuat apakah program yang akan kita buat itu efektif dan efisien tidak.

Menghitung Kompleksitas Waktu

This is the most important part from this slide
Kalo ga ada ini, palingan gaada yang mau baca :v

Penjelasan Awal

Dalam kompleksitas, seringkali terdapat penggunaan variable (n, m) yang berarti kompleksitas program tersebut bergantung dengan banyaknya inputan tersebut.

Misalkan kompleksitas bubble sort $O(n^2)$, dimana n merupakan jumlah data. Berarti jika datanya ada 10, maka kompleksitasnya $\mathrm{O}(10^2 = 100)$ dan jika datanya ada $10^6(1\ juta)$ maka kompleksitasnya $O(10^{12})$.

Terminologi

Best case (notasinya Ω omega) Untuk menunjukkan waktu tercepat program kita berjalan

Average case (notasinya Θ theta) Untuk menunjukkan rata-rata waktu program kita berjalan

Worst case (notasinya O big oh) Untuk menunjukkan waktu terlambat program kita berjalan

Karena kita harus selalu mempersiapkan yang terburuk, kita akan banyak sekali membahas dalam worst case (big oh)

Waktu

Biasanya, menurut *rule thumb* yang berlaku, untuk setiap $O(100.000.000\ atau\ 10^8)$ itu berjalan selama 1 detik.

Sehingga jika ada program dengan kompleksitas $O(1.000.000.000\ atau\ 10^9)$ maka program akan berjalan selama $O(10\ \times 10^8) = 10\ O(10^8) = 10\ detik$

Jika ada program dengan kompleksitas $O(1.000.000\ atau\ 10^6)$ maka program akan berjalan selama $O(10^{-2}\ \times 10^8) = 10^{-2}\ O(10^8) = 10^{-2}\ detik = 0.01\ detik$

Big Oh

Dalam Big Oh, terdapat aturan yang memudahkan penghitungan:

1. Jika $O(f(x))$ merupakan penjumlahan dari beberapa variable, maka variable yang sama dengan "pertumbuhan" terkecil dapat diabaikan

Misalkan

$$O(n^3 + 3n^2 + n) = O(n^3)$$

$3n^2 + n$ dapat diabaikan karena memiliki variable yang sama dengan n^3. Jika n nya cukup besar, maka yang memberikan "nilai terbesar" adalah n^3

Tetapi

$$O(n^2 \log n + m) = O(n^2 \log n + m)$$

Tetap ditulis karena $n^2 \log n$ merupakan perkalian. Nilai m dan n tidak memiliki hubungan

Kecuali,

jika diberitahu bahwa $m = n^2$

maka kompleksitasnya menjadi

$$O(n^2 \log n + m) = O(n^2 \log n + n^2) = O(n^2 \log n)$$

Big Oh

2. Jika $O(f(x))$ mengandung koefisien, maka koefisiennya dapat diabaikan

Misalkan

$$O(2n^3 + 5n) = O(2n^3) = O(n^3)$$

Penjelasan Lanjut

Big Oh memberikan informasi mengenai program dapat selesai eksekusi paling banyak O(x) kali.

Paling banyak berarti tidak akan ada proses yang lebih banyak dibandingkan nilai big oh.

Misalkan

Terdapat program dengan kompleksitas O(n). Kita bisa mengatakan program tersebut O(n^2), karena proses yang dibutuhkan tidak akan melebihi n^2. Tetapi akan lebih tepat jika mengatakan prosesnya O(n)

Kemungkinan, soal kompleksitas ini akan memiliki beberapa jawaban yang beragam. Tetapi tetap ada jawaban yang paling tepat.

Pertumbuhan Kompleksitas

Semakin ke kanan, semakin besar

Kompleksitas	$O(1)$	$O(loglog(n))$	$O(\log n)$	$O(\sqrt[x]{n})$ x = ..,3,2	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^k)$, k = 3,4,.	$O(2^n)$	$O(n!)$
Besar n agar Big Oh 10^8	∞	$2^{2^{10^8}}$	2^{10^8}	10^{8x}	10^8	$\sim 5 \times 10^6$	10^4	$10^{\frac{8}{k}}$	27	11

Jadi jika ada kompleksitas $O(n^3 + n \lg n + 2^n)$ dapat disederhanakan menjadi $O(2^n)$, karena 2^n memiliki pertumbuhan terbesar (posisi table di paling kanan daripada yang lain)

Cara biar gausa inget, masukin aja n yang cukup besar. Trus liat yang paling besar yang mana

F.A.Q

Kapan pakai $\lg n$, $\log n$, $\ln n$?

Sebenarnya dalam kompleksitas, semuanya bisa dianggap sama

$$\lg n = \log_2 n = \frac{\log n}{\log 2} = \log n$$

Karena $\log 2$ adalah konstanta. Begitu juga untuk $\ln n$

Tapi jika memang polanya adalah $harmonic\ series$, langsung pakai $\ln n$ aja.

Notes: $\lg n = \log_2 n$

Notes: $\ln n = \log_e n$, e = bilangan natural (2.718...)

Beberapa formula yang perlu diingat (bisa diturunkan sih ☺)

$$1 + 2 + 3 + \ \ldots + n = \frac{n(n+1)}{2}$$

$$a + (a + b) + (a + 2b) + \cdots + (a + (n-1)b) = \frac{n}{2}(2a + (n-1)b) \ \textit{(deret aritmatika)}$$

$$a + ar + ar^2 + \cdots + ar^{n-1} = \frac{a(r^n - 1)}{r-1}, r > 1 \ \textit{(deret geometri)}$$

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n \ \ (harmonic\ series)$$

$$\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} \ \ldots + \frac{1}{p} = \sum_{i=2}^{n, p_i\ prima} \frac{1}{i} \approx \ln \ln n \ \ (sum\ of\ reciprocal\ prime)$$

$$a^{\log_a b} = b$$

Bagaimana cara menghitung kompleksitas?

1. Lakukan analisa yang dibutuhkan. Apa saja yang perlu dihitung?
2. Lakukan penjabaran kompleksitas secara menyeluruh.
3. Ambil pertumbuhan terbesar dari variable

Jika pusing, lakukan rough estimation aja

Bingung? Coba contoh soal deh

Contoh 1

```
for(int i=1;i<=n;i++){
//Pada umumnya, jika di bagian ini tidak melakukan apa2, maka bisa dianggap O(1)
}
```

Bagian dalam for akan dijalankan sebanyak n kali. Sehingga kompleksitasnya bisa disebut O(n)

Contoh 2

```
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
      //Proses dalam O(1)
    }
}
```

Lakukan penjabaran.

Jika i = 1, j akan dijalankan dari 1 → n O(n)

Jika i = 2, j akan dijalankan dari 1 → n O(n)

...

Jika i = n, j akan dijalankan dari 1→ n O(n)

Maka O(n) akan dilakukan sebanyak n kali. Sehingga kompleksitasnya menjadi O(n*n = n^2)

Contoh 3

```
for(int i=1;i<=n;i++){
    for(int j=i;j<=n;j++){
      //Proses dalam O(1)
    }
}
```

Lakukan penjabaran.

Jika i = 1, j akan dijalankan dari 1 → n O(n)

Jika i = 2, j akan dijalankan dari 2 → n O(n-1)

…

Jika i = n, j akan dijalankan dari n→ n O(1)

Maka total kompleksitasnya adalah O(n + (n-1) + … + 1) = O($\frac{n(n+1)}{2}$) = O($\frac{(n^2+n)}{2}$) = O(n^2)

Contoh 4

```
for(int i=1;i<=n;i++){
    for(int j=i;j<=n;j+=i){
        //Proses dalam O(1)
    }
}
```

Lakukan penjabaran.

Jika i = 1, j akan dijalankan dari 1 → n O(n)

Jika i = 2, j akan dijalankan dari 2 → n, dengan rough estimation O(n/2)

…

Jika i = n, j akan dijalankan dari n→ n O(1)

Maka total kompleksitasnya adalah O(n + n/2+ … + 1) = O($n(\frac{1}{1} + \ \ldots + \frac{1}{n})$) = O($n \ln n$)

Contoh 5

Set num to 0.

num[1] = 1

```
for(int i=2;i<=n;i++){
    If(num[i] == 0){
        for(int j=i;j<=n;j+=i){
            num[j] = I;
        }

    }
}
```

Lakukan penjabaran.

Baris paling atas O(n)

Baris selanjutnya O(1)

Bagian dalam perulangan

Jika i = 2, j akan dijalankan dari 2 → n, dengan rough estimation O(n/2)

Jika i = 3, j akan dijalankan dari 3 → n, dengan rough estimation O(n/3)

Jika i = 4, j tidak akan dijalankan, O(1)

Jika i = 5, j akan dijalankan dari 5 → n, dengan rough estimation O(n/5)

Jika i = 6, j tidak akan dijalankan, O(1)

Bagian perulangan berjalan sebesar $O\left(\frac{n}{2}+\frac{n}{3}+1+\frac{n}{5}+1+\frac{n}{7}+\ldots\right)=$

$O\left(\frac{n}{2}+\frac{n}{3}+\frac{n}{5}+\frac{n}{7}+\ \ldots+\frac{n}{p}+k(bilangan\ non\ prima)\right)=$ Jika kita mengeluarkan n, maka n(1/2 + 1/3+1/5+…) ← sum of reciprocal prime=

$O(nln\ln n+k)=O(n\ln\ln n)$

Contoh 6

```
getline(cin,input);
for(int i=0;i<strlen(input);i++){
    printf(“Karakter %d adalah %c”,i,input[i]);
}
```

Berapa besar kompleksitasnya?

Uda lama gaada gambar ☹

Weit, what?

Meskipun sekilas kompleksitasnya O(|N|), tetapi ternyata tidak seperti demikian.

```
getline(cin,input);
for(int i=0;i<strlen(input);i++){
    printf(“Karakter %d adalah %c”,i,input[i]);
}
```

Coba kita Analisa:

Getline // O(N)

for dijalankan dari 0 ->N-1 O(N).

Tetapi terdapat kompleksitas rahasia, yaitu ketika membandingkan i dengan strlen(input), kita secara tidak sadar akan melakukan O(N) setiap pemanggilannya. Fungsi strlen input akan dipanggil sebanyak panjang input. Terkadang ada beberapa fungsi dalam C/C++ yang memiliki besar kompleksitas sendiri. Kita harus memperhatikan hal tersebut.

Sehingga kompleksitas di atas sebesar O(N*N = N^2). Bagaimana cara mempercepatnya?

Hitung sekali saja di awal.

```
getline(cin,input);
int len = strlen(input);
for(int i=0;i<len;i++){
   printf("Karakter %d adalah %c",i,input[i]);
}
```

Sehingga kompleksitasnya menjadi:

```
getline(cin,input); //O(N)
int len = strlen(input); //O(N)
```

for dijalankan dari 0 ->N-1 O(N).

Kompleksitasnya menjadi O(N+N+N = N)

Kompleksitas Tersembunyi

So, apa saja yang memiliki kompleksitas tersembunyi?

String manipulation→ strlen,strcat,strcpy,strstr = O(|Panjang String|)
Math power → pow = O(llog besar pangkat)
Math trigonometry → cos,sin,tan,acos,asin,atan = O(log besar angka)
Math root → sqrt,cbrt = O(log besar angka)
STL function → insert, find = O(log banyakdata)
Tapi untuk length di class string O(1)

Pokoknya fungsi yang memudahkan tuh, pasti ada harganya ☺
So hati-hati dalam pakai fungsi.

Contoh soal 7

```
int lo = 1;
int hi = n;
while(lo <= hi){
    int mid = (lo + hi)/2;
    for(int i=1;i<=n;i++){
      //O(1)
    }
    for(int j=1;j*j<=m;j++){
      //O(1)
    }
    hi = mid – 1;
}
```

Berapa kompleksitas kodingan mengerikan ini?

Di dalam pengulangan while, akan berjalan sebanyak:

1. for(int i=1;i<=n;i++), sebesar O(n)
2. for(int j=1;j*j<=m;j++), sebesar O($\sqrt{m}$), karena kita hanya perlu melakukan pengulangan sebanyak j kali dimana $j^2 \leq m$. Jika kita akarkan kedua ruas, didapat $j \leq \sqrt{m}$

Sehingga kompleksitas dalam while menjadi $O(n + \sqrt{m})$.

Lalu bagaimana dengan whilenya?

Karena tiap pengulangan kita "membuang" setengah dari nilai n sekarang, maka bisa kita tulis seperti ini:

$n * \frac{1}{2} * \frac{1}{2} * \cdots * \frac{1}{2} (sebanyak\ iterasi, sebut\ saja\ k) = 1$

$n\left(\frac{1}{2}\right)^k = 1$, lakukan kali silang

$n = 2^k$, lakukan logaritma

$k = \lg n$

Jadi karena setiap while berjalan $O(n + \sqrt{m})$, dan terdapat $\lg n$ pengulangan while, maka kompleksitasnya menjadi $O(n \lg n + \sqrt{m} \lg n)$.

Here is an image of
beautiful girl
to relax your mind

Kompleksitas Rekursif

Sebenarnya kompleksitas rekursif “hampir sama” dengan kompleksitas pada umumnya. Yang menjadi permasalahan utama adalah, seberapa dalam rekursif tersebut akan dipanggil, dan setiap pemanggilannya memakan kompleksitas berapa?

Contoh soal

```
 int pangkat5(int x){
    If(x == 0) return 1;
    return 5 * pangkat5(x-1);
}
```

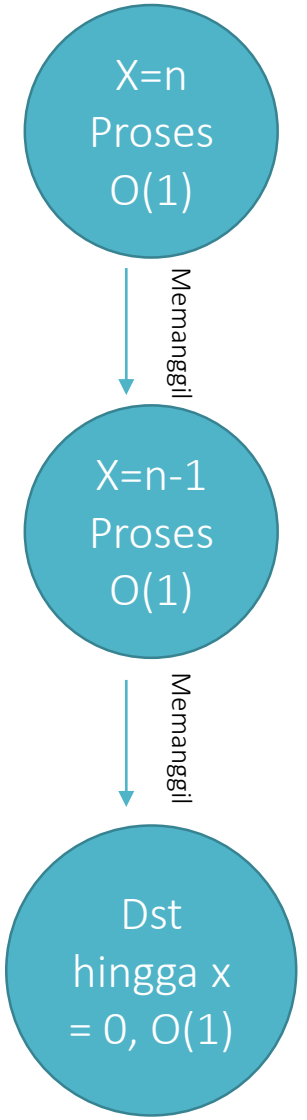
Jika dipanggil pangkat5(n), maka berapa besar kompleksitasnya?

Solusi

Kita bisa melihatnya dalam bentuk gambar

Jika didefinisikan, setiap lingkaran adalah pemanggilan fungsi. Dalam sebuah rekursi, akan terdapat n pemanggilan fungsi dimana setiap pemanggilan fungsi adalah O(1)

Sehingga kompleksitasnya O(1 * n = n)



Contoh soal

```
 int pangkat5(int x){
    If(x == 0) return 1;
    return 2 * pangkat5(x-1) + 3 * pangkat5(x-1);
}
```

Jika dipanggil pangkat5(n), maka berapa besar kompleksitasnya?

Solusi

~~2 * pangkat5(x-1) + 3 * pangkat5(x-1) = 5 * pangkat5(x-1)~~

~~Jadi seperti soal sebelumnya~~

Sebuah fungsi (dalam Bahasa C/C++) tidak dapat melakukan penggabungan fungsi. Sehingga kompleksitasnya akan berbeda

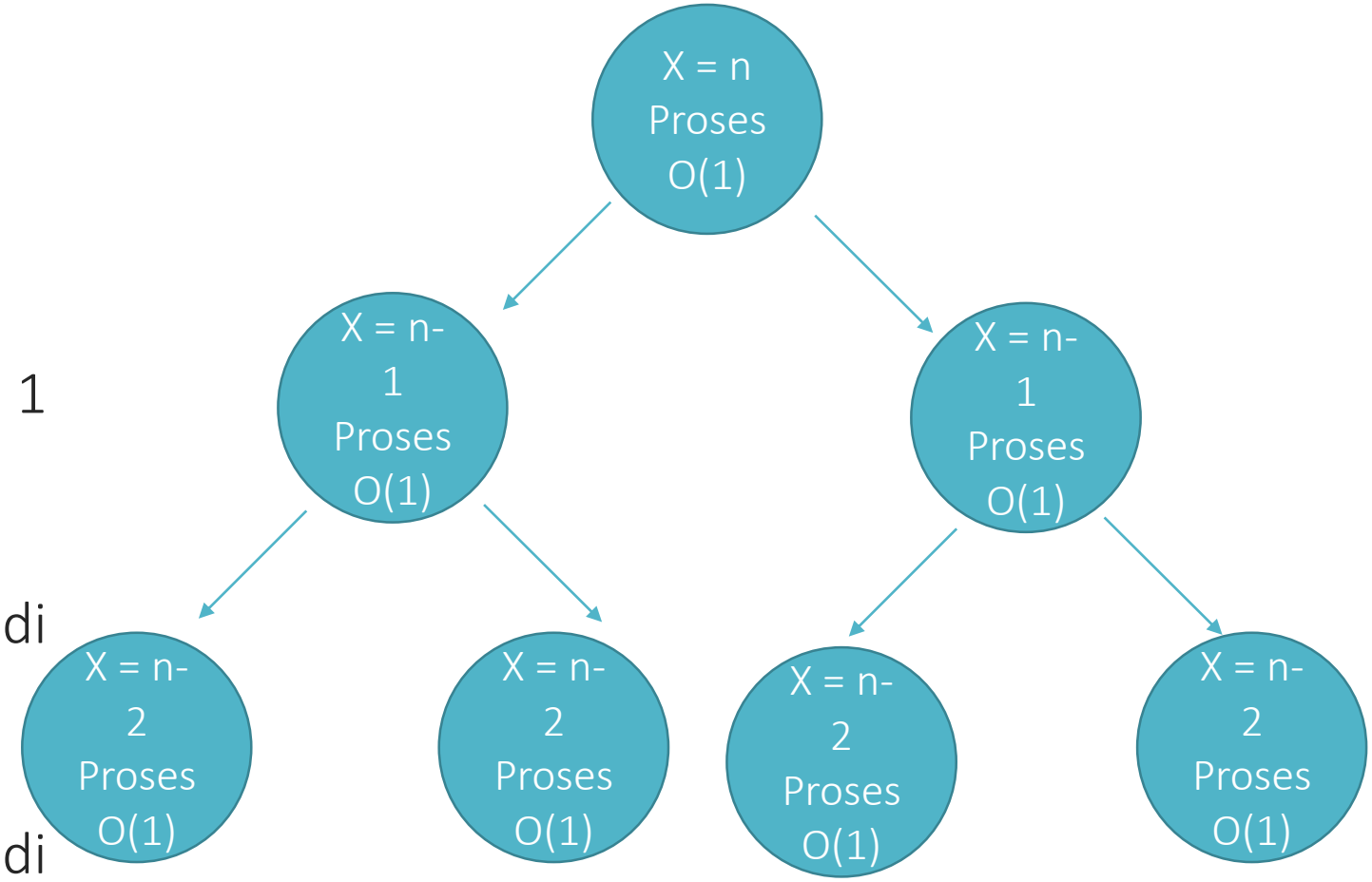
Solusi

Jika digambar akan sebagai berikut

Untuk x = n (tingkat 1) akan terjadi 1 = (2^0) pemanggilan fungsi

Untuk x = n-1 (tingkat 2) akan terjadi 2 = (2^1) pemanggilan fungsi

Untuk x = n-2 (tingkat 2) akan terjadi 2 kali lebih banyak dibandingkan sebelumnya, yaitu 4 = (2^2)



Solusi

Untuk tingkat ke i, akan memiliki 2^(i-1) pemanggilan.

Untuk setiap pemanggilan, akan menghasilkan kompleksitas sebesar O(1)

Sehingga kompleksitasnya sebesar:

$O(2^0 + 2^1 + 2^2 + \ldots + 2^{(n-1)})*O(1)$

Dengan menggunakan deret geometri

$O(2^n - 1)*O(1) = O(2^n)$

Contoh Soal

```
 int fibo(int x){
    If(x <= 1) return 1;
    return fibo(x-1) + fibo(x-2);
}
```

Jika dipanggil fibo(n), maka berapa besar kompleksitasnya?

Solusi

Jika digambar, setiap bagian tentunya akan memiliki tingkat yang berbeda namun tidak terlalu jauh bedanya. Kita bisa menyamakan sehingga memiliki tingkatan yang sama

Sehingga kompleksitasnya adalah O(2^n)

Contoh Soal

```
 int query(int idx,int left,int right){
    if(left == right) return arr[idx];
    int mid = (left + right)/2;
    return query(2*idx,left,mid) + query(2*idx+1,mid+1,right);
}
```

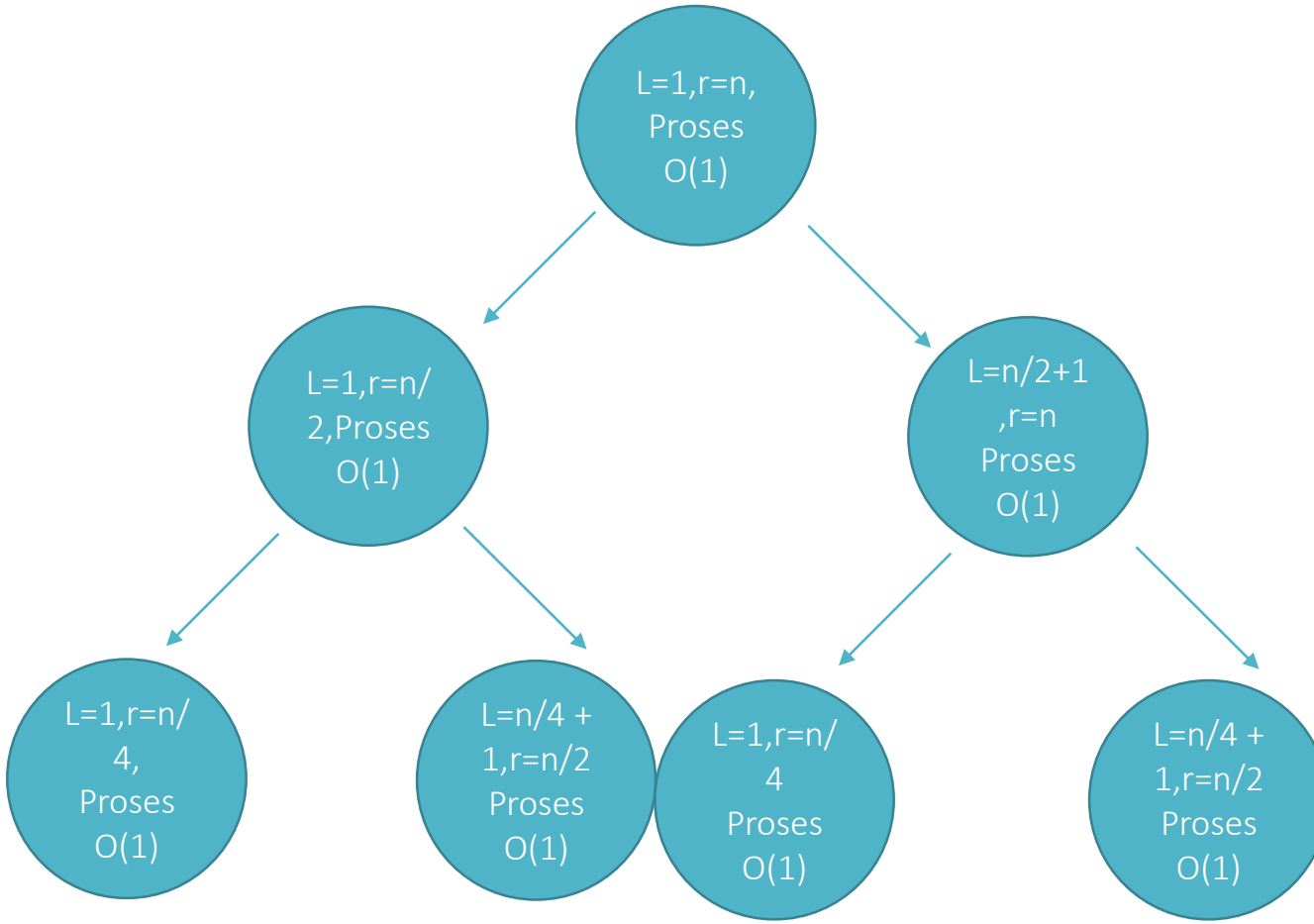
Jika dipanggil fungsi query(1,1,n). Tentukan besar kompleksitas program tersebut

L = left, r = right

Solusi

Perhatikan gambar di samping

Karena tiap pengulangan kita "membuang" setengah dari nilai n sekarang, maka tree ini maksimal memiliki ketinggian lg n.



Untuk tingkat ke i akan menghasilkan node sebanyak 2^(i-1)

Sehingga kompleksitas akhirnya adalah:

$O(2^0 + 2^1 + 2^2 + \ldots + 2^{(\lg n-1)})*O(1)$

$O(2^{\lg n} - 1)) = O(2^{\lg n}) = O(n)$ //lg n adalah log basis 2

Contoh Soal

```
 int query(int idx,int left,int right,int target){
    if(left == right && left == target) return arr[idx];
    if(left > target || right < target) return 0;
    int mid = (left + right)/2;
    return query(2*idx,left,mid,target) + query(2*idx+1,mid+1,right,target);
}
```

Jika dipanggil fungsi query(1,1,n,x). Tentukan besar kompleksitas program tersebut

L = left, r = right, t = target

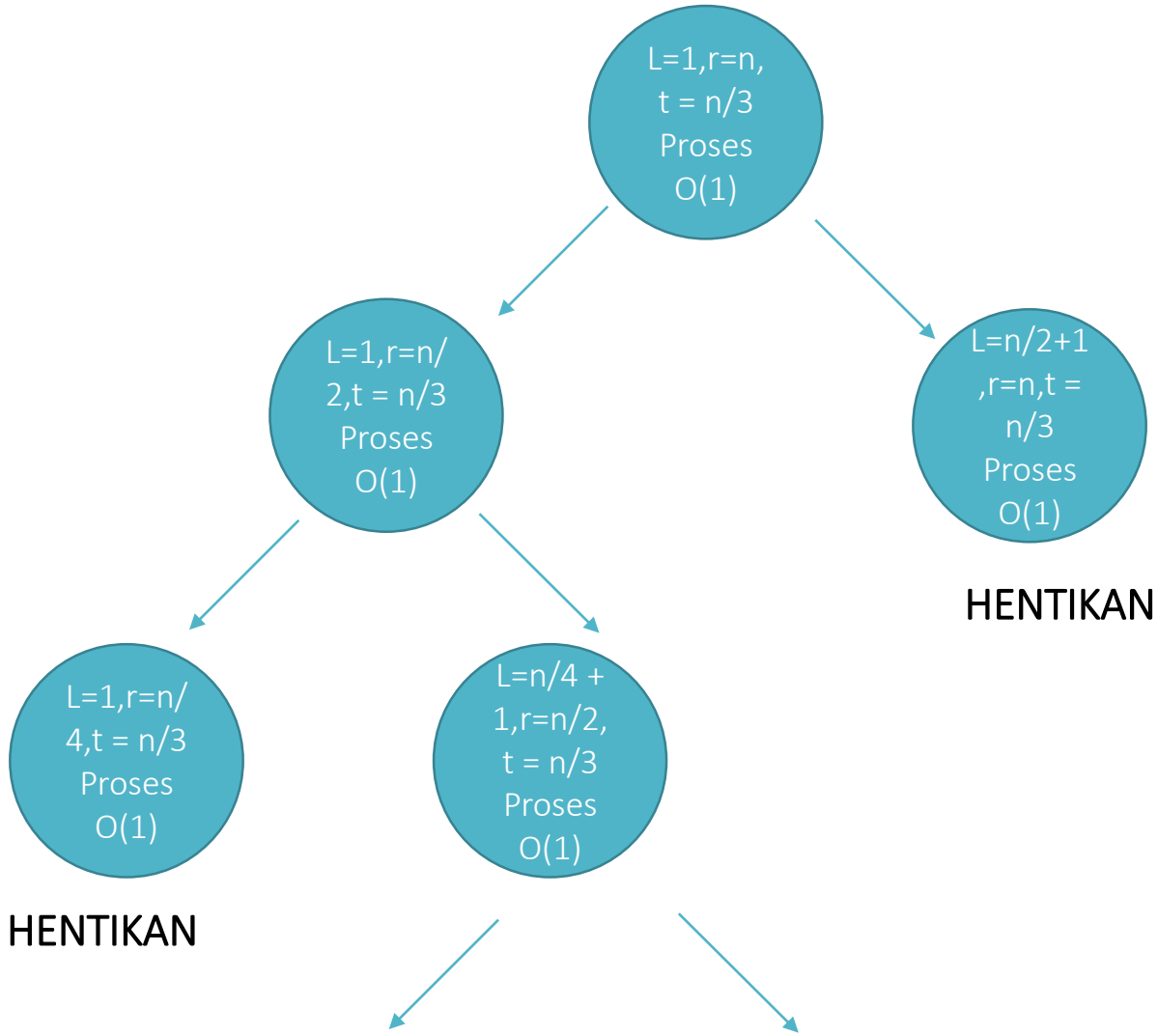
Solusi

Perhatikan gambar di samping

Meskipun akan memanggil 2 fungsi untuk setiap pemanggilan, tetapi terdapat 1 fungsi yang tidak terpakai (hanya dipanggil sekali) dan tidak melakukan rekursif lagi.

Karena tiap pengulangan kita “membuang” setengah dari nilai n sekarang, maka tree ini maksimal memiliki ketinggian log n.

Sehingga kompleksitasnya O(log n)



Contoh Soal

```
 void query(int left,int right){
    if(left == right) return ;
    For(int i=left;i<=right;i++) printf(“%d ”,arr[i]);
    int mid = (left + right)/2;
    query(left,mid);
    query(mid+1,right);
}
```

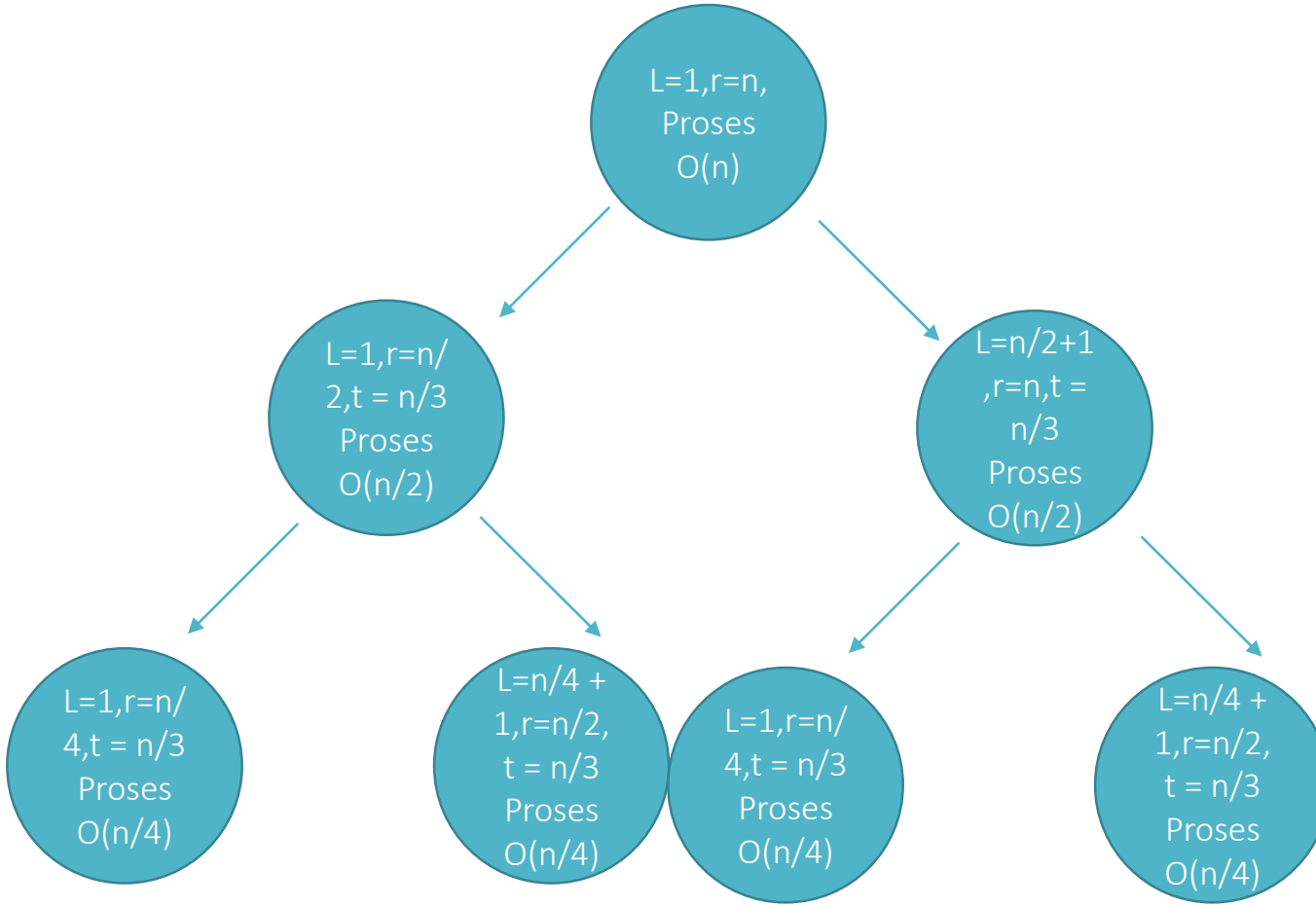
Jika dipanggil fungsi query(1,n). Tentukan besar kompleksitas program tersebut

L = left, r = right

Solusi

Perhatikan gambar di samping

Karena tiap pengulangan kita "membuang" setengah dari nilai n sekarang, maka tree ini maksimal memiliki ketinggian lg n.



Untuk tingkat ke I, akan menghasilkan kompleksitas total sebanyak O(n) (diserahkan untuk latihan)

Sehingga kompleksitas akhirnya adalah:

O(n lg n)

END OF FILE