CPS 616: Algorithms

Asymptotic Notation - Part I

January 18, 2022

Onur Çağırıcı

Outline

- Understanding of big-Oh
- Formal definition of big-Oh
- Omega notation
- Theta notation
- Little-Oh
- Little omega

Read Chapter 2.2 and 3.1 of the book

Comparing two programs for one problem

Alice:

- Finding the smallest element in an array
- Her algorithm takes 910 milliseconds to run when an array of 100 is given
- Her system: Linux, 8Gb RAM
- Programming language: Python

Bob:

- Finding the smallest element in an array
- His algorithm takes 1050 milliseconds to run when an array of 90 is given
- His system: Windows 10, 16Gb RAM
- Programming language: C#

Comparing two programs for one problem

Alice:

- For the input of size n, the minimum number in the array can be found with $f(n)$ primitive operation:

$$f(n) = 12n \log n + 13n + 500$$
$$f(n) \in \Theta(n \log n)$$

Bob:

- For the input of size m, the minimum number in the array can be found with $g(m)$ primitive operation:

$$g(m) = 2000m$$
$$g(m) \in \Theta(m)$$

Asymptotic notations

- Expressing the time complexity better
- We can compare easier
  - O (big-Oh)
  - Θ (theta)
  - Ω (omega)
  - o (little-Oh)
  - ω (little omega)

Asymptotic notations

1. Find the function that the cost of the algorithm is based on

Alice's algorithm:

1. $f(n) =$
2. $f(n) \in$

Asymptotic notations

1. Find the function that the cost of the algorithm is based on

Alice's algorithm:

1. $f(n) = 12n \log n + 13n + 500$
2. $f(n) \in$

Asymptotic notations

1. Find the function that the cost of the algorithm is based on
2. Express it by asymptotic notations

Alice's algorithm:

1. $f(n) = 12n \log n + 13n + 500$
2. $f(n) \in$

Asymptotic notations

1. Find the function that the cost of the algorithm is based on
2. Express it by asymptotic notations

Alice's algorithm:

1. $f(n) = 12n \log n + 13n + 500$
2. $f(n) \in \Theta(n \log n)$

Analyzing the running time of a code

Algorithm: $\textsc{FindMin}(A)$

Analyzing the running time of a code

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

Analyzing the running time of a code

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

$min \leftarrow A[0]$;

$i \leftarrow 0$;

while $i < n$ **do**

  if $A[i] < min$ **then**

    $min \leftarrow A[i]$;

  $i \leftarrow i + 1$;

return min

Analyzing the running time of a code

- Time
- Space
- Network traffic
- Time of the developer to program
- Complexity of the code

For us and in this course: running time

Expression based on the size of the input

- Running the program on 1 item is not the same as running the program on 10^7 items.

Expression based on the size of the input

- Running the program on 1 item is not the same as running the program on 10^7 items.
- Finding the minimum element in the array of size 1 is different than finding the minimum element in an array of size 10^7.

Expression based on the size of the input

- Running the program on 1 item is not the same as running the program on 10^7 items.
- Finding the minimum element in the array of size 1 is different than finding the minimum element in an array of size 10^7.

$$f(n) = 12n\log n + 13n + 500$$
$$g(n) = 2000n$$

Word RAM model

- Primitive operations

Word RAM model

- Primitive operations
  - Basic mathematics: Addition, subtraction, multiplication, division
  - Logistic operations: AND, OR, XOR, NOT, and bit shift on words
  - Boolean operation: $<, >, ==, != , \geq, \leq$
  - Read or write a word

Word RAM model

- Primitive operations
  - Basic mathematics: Addition, subtraction, multiplication, division
  - Logistic operations: AND, OR, XOR, NOT, and bit shift on words
  - Boolean operation: $<, >, ==, != , \geq, \leq$
  - Read or write a word
- Non-primitive operations

Word RAM model

- Primitive operations
  - Basic mathematics: Addition, subtraction, multiplication, division
  - Logistic operations: AND, OR, XOR, NOT, and bit shift on words
  - Boolean operation: $<, >, ==, != , \geq, \leq$
  - Read or write a word
- Non-primitive operations
  - Exponentiation
  - Logarithms

Running time

- Code is a set of instructions
- Each instruction takes one clock cycle to run
- Each of these instructions is called a primitive operation
  - We must count the number of primitive operations
  - In code: count the number of steps

What is a step of a code?

Usually a line of code without a loop or a method call $\rightarrow O(1)$ time.

What is a step of a code?

Usually a line of code without a loop or a method call $\rightarrow O(1)$ time.

An example code with 5 steps:

```
int i;
int j;
j = 0;
i = 23;
i = i*j;
```

What is a step of a code?

Usually a line of code without a loop or a method call → $O(1)$ time.

An example code with 5 steps:

```
int i;
int j;
j = 0;
i = 23;
i = i*j;
```

The same operation in 3 steps:

```
int i = 23;
int j = 0;
i = i*j;
```

What is a step of a code?

Usually a line of code without a loop or a method call $\rightarrow O(1)$ time.

An example code with 5 steps:

```
int i;
int j;
j = 0;
i = 23;
i = i*j;
```

The same operation in 3 steps:

```
int i = 23;
int j = 0;
i = i*j;
```

- We want a function to express running time proportional to the number of primitive operations.
- Counting the steps and using asymptotic notation are much easier than counting the primitive operations.

Reminder

- Find a function representing the running time of the code
- Remember that the size of input is not predictable

What are we looking for in running time?

- Best-case time
- Worst-case time
- Average-case time

Worst-case running time means that the given input is a nightmare for an algorithm. Slowest behavior of the algorithm and gives us an upper bound on the time complexity.

Best-case running time means that the given input is almost immediately solvable by the algorithm. Fastest behavior gives us a lower bound on the time complexity.

Average-case running time of an algorithm is the algorithm's behavior averaged over all possible inputs.

Example: Find the minimum number in an array

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

$min \leftarrow A[0]$;

$i \leftarrow 0$;

while $i < n$ **do**

  if $A[i] < min$ **then**

    $min \leftarrow A[i]$;

  $i \leftarrow i + 1$;

return min

$$f(n) =$$

Example: Find the minimum number in an array

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

$min \leftarrow A[0]$;

$i \leftarrow 0$;

while $i < n$ **do**

    if $A[i] < min$ **then**

        $min \leftarrow A[i]$;

    $i \leftarrow i + 1$;

return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

Example: Find the minimum number in an array

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers
Output: Minimum integer in A

$min \leftarrow A[0]$;
$i \leftarrow 0$;
while $i < n$ **do**
  if $A[i] < min$ **then**
    $min \leftarrow A[i]$;
  $i \leftarrow i + 1$;
return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

Example: Find the minimum number in an array

Algorithm: FindMin(A)

Input: An array A of n integers

Output: Minimum integer in A

```
min ← A[0];
i ← 0;
while i < n do
    if A[i] < min then
        min ← A[i];
    i ← i + 1;
return min
```

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

Example: Find the minimum number in an array

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

$min \leftarrow A[0]$;

$i \leftarrow 0$;

while $i < n$ **do**

  if $A[i] < min$ **then**

    $min \leftarrow A[i]$;

  $i \leftarrow i + 1$;

return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

Example: Find the minimum number in an array

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

$min \leftarrow A[0]$;

$i \leftarrow 0$;

while $i < n$ **do**

  if $A[i] < min$ **then**

    $min \leftarrow A[i]$;

  $i \leftarrow i + 1$;

return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

Example: Find the minimum number in an array

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers

Output: Minimum integer in A

$min \leftarrow A[0]$;

$i \leftarrow 0$;

while $i < n$ **do**

  if $A[i] < min$ **then**

    $min \leftarrow A[i]$;

  $i \leftarrow i + 1$;

return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

$$= 1 + 3n$$

Another example

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers
Output: ?

$n \leftarrow A.length$;
for $i = 0 \rightarrow n - 2$ **do**
    for $j = i + 1 \rightarrow n - 1$ **do**
        if $A[i] > A[j]$ **then**
            $temp \leftarrow A[i]$;
            $A[i] \leftarrow A[j]$;
            $A[j] \leftarrow temp$;

Another example

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers
Output: ?

```
n ← A.length;
for i = 0 → n − 2 do
    for j = i + 1 → n − 1 do
        if A[i] > A[j] then
            temp ← A[i];
            A[i] ← A[j];
            A[j] ← temp;
```

- What does this function do?

Another example

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers
Output: **?**

$n \leftarrow A.length$;
for $i = 0 \rightarrow n-2$ **do**
  for $j = i+1 \rightarrow n-1$ **do**
    if $A[i] > A[j]$ **then**
      $temp \leftarrow A[i]$;
      $A[i] \leftarrow A[j]$;
      $A[j] \leftarrow temp$;

- What does this function do?
- What is the worst-case running time?

Selection sort

Algorithm: $\mathrm{F1}(A)$

$f(n) =$

Input: An array A of n integers
Output: ?

$n \leftarrow A.length$;
for $i = 0 \rightarrow n - 2$ **do**
  for $j = i + 1 \rightarrow n - 1$ **do**
    if $A[i] > A[j]$ **then**
      $temp \leftarrow A[i]$;
      $A[i] \leftarrow A[j]$;
      $A[j] \leftarrow temp$;

Selection sort

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers
Output: ?

$n \leftarrow A.length;$
for $i = 0 \rightarrow n - 2$ **do**
    for $j = i + 1 \rightarrow n - 1$ **do**
        if $A[i] > A[j]$ **then**
            $temp \leftarrow A[i];$
            $A[i] \leftarrow A[j];$
            $A[j] \leftarrow temp;$

$f(n) =$

Selection sort

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers

Output: ?

$n \leftarrow A.length;$

for $i = 0 \rightarrow n - 2$ **do**

  for $j = i + 1 \rightarrow n - 1$ **do**

    if $A[i] > A[j]$ **then**

      $temp \leftarrow A[i];$

      $A[i] \leftarrow A[j];$

      $A[j] \leftarrow temp;$

$$f(n) = 1 + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 4$$

Selection sort

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers
Output: ?

$n \leftarrow A.length;$
for $i = 0 \rightarrow n-2$ **do**
  for $j = i+1 \rightarrow n-1$ **do**
    if $A[i] > A[j]$ **then**
      $temp \leftarrow A[i];$
      $A[i] \leftarrow A[j];$
      $A[j] \leftarrow temp;$

$$f(n) = 1 + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 4$$

$$= 1 + \sum_{i=0}^{n-2} 4(n-i-1)$$

Selection sort

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers

Output: ?

$n \leftarrow A.length;$

for $i = 0 \to n-2$ **do**

  for $j = i+1 \to n-1$ **do**

    if $A[i] > A[j]$ **then**

      $temp \leftarrow A[i];$

      $A[i] \leftarrow A[j];$

      $A[j] \leftarrow temp;$

$$
\begin{aligned}
f(n) &= 1 + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 4 \\
&= 1 + \sum_{i=0}^{n-2} 4(n-i-1) \\
&= 1 + \sum_{i=0}^{n-2} 4n - \sum_{i=0}^{n-2} 4i - \sum_{i=0}^{n-2} 4 \\
&= 1 + 4n(n-1) - \frac{4(n-2)(n-1)}{2} - 4(n-1)
\end{aligned}
$$

Selection sort

Algorithm: $\mathrm{F1}(A)$

Input: An array A of n integers
Output: ?

$n \leftarrow A.length;$
for $i = 0 \to n-2$ **do**
  for $j = i+1 \to n-1$ **do**
    if $A[i] > A[j]$ **then**
      $temp \leftarrow A[i];$
      $A[i] \leftarrow A[j];$
      $A[j] \leftarrow temp;$

$$
\begin{aligned}
f(n) &= 1 + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 4 \\
&= 1 + \sum_{i=0}^{n-2} 4(n-i-1) \\
&= 1 + \sum_{i=0}^{n-2} 4n - \sum_{i=0}^{n-2} 4i - \sum_{i=0}^{n-2} 4 \\
&= 1 + 4n(n-1) - \frac{4(n-2)(n-1)}{2} - 4(n-1) \\
&= 2n^2 - 2n + 1
\end{aligned}
$$

Running time of mathematical functions in CPU when $n = 1000$

Assuming CPU has the power to run one million instructions per second

1	constant	1 microsecond
$\log n$	logarithmic	6.9 microseconds
$\sqrt{n}$	sublinear	31 microseconds
n	linear	1 millisecond
$n \log n$	linearithmic	6.9 milliseconds
n^2	quadratic	1 second
n^3	cubic	16 minutes
n^4	quartic	11 days
2^n	exponential	3.4×10^{287} years
$n!$	factorial	1.3×10^{2554} years

Why big-Oh?

Why big-Oh?

It is a simple way for expressing the running time.

Why big-Oh?

It is a simple way for expressing the running time.

$$f(n) = \begin{cases} (n \log n)^4 + \sqrt{n} + 12 & , n > 20 \\ n + \sqrt{\log n} & , 10 < n < 20 \\ n^7 \log n & , n < 10 \end{cases}$$

Why big-Oh?

It is a simple way for expressing the running time.

$$f(n) = \begin{cases} (n \log n)^4 + \sqrt{n} + 12 & , n > 20 \\ n + \sqrt{\log n} & , 10 < n < 20 \\ n^7 \log n & , n < 10 \end{cases}$$

$$f(n) \in O(n^7 \log n)$$

Algorithms finding the minimum number in an array with different complexities

Algorithm: Alice's algorithm

$n \leftarrow A.length$;
for $i = 0 \rightarrow n-2$ **do**
    for $j = i+1 \rightarrow n-1$ **do**
        if $A[i] > A[j]$ **then**
            $temp \leftarrow A[i]$;
            $A[i] \leftarrow A[j]$;
            $A[j] \leftarrow temp$;

$min \leftarrow A[0]$;
for $j = i+1 \rightarrow n-1$ **do**
    if $A[i] < min$ **then**
        $min \leftarrow A[i]$;

return min

Algorithm: Bob's algorithm

$n \leftarrow A.length$;
$min \leftarrow A[0]$;
for $i = 0 \rightarrow n-1$ **do**
    if $A[i] < min$ **then**
        $min \leftarrow A[i]$;

return min

Alice's algorithm: $f(n) \in$

Bob's algorithm: $g(n) \in$

Algorithms finding the minimum number in an array with different complexities

Algorithm: Alice's algorithm

$n \leftarrow A.length$;
for $i = 0 \rightarrow n-2$ **do**
  for $j = i+1 \rightarrow n-1$ **do**
    if $A[i] > A[j]$ **then**
      $temp \leftarrow A[i]$;
      $A[i] \leftarrow A[j]$;
      $A[j] \leftarrow temp$;
$min \leftarrow A[0]$;
for $j = i+1 \rightarrow n-1$ **do**
  if $A[i] < min$ **then**
    $min \leftarrow A[i]$;
return min

Algorithm: Bob's algorithm

$n \leftarrow A.length$;
$min \leftarrow A[0]$;
for $i = 0 \rightarrow n-1$ **do**
  if $A[i] < min$ **then**
    $min \leftarrow A[i]$;
return min

Alice's algorithm: $f(n) \in O(n^2)$
Bob's algorithm: $g(n) \in$

Algorithms finding the minimum number in an array with different complexities

Algorithm: Alice's algorithm

$n \leftarrow A.length$;
for $i = 0 \rightarrow n-2$ **do**
    for $j = i+1 \rightarrow n-1$ **do**
        if $A[i] > A[j]$ **then**
            $temp \leftarrow A[i]$;
            $A[i] \leftarrow A[j]$;
            $A[j] \leftarrow temp$;
$min \leftarrow A[0]$;
for $j = i+1 \rightarrow n-1$ **do**
    if $A[i] < min$ **then**
        $min \leftarrow A[i]$;
return min

Algorithm: Bob's algorithm

$n \leftarrow A.length$;
$min \leftarrow A[0]$;
for $i = 0 \rightarrow n-1$ **do**
    if $A[i] < min$ **then**
        $min \leftarrow A[i]$;
return min

Alice's algorithm: $f(n) \in O(n^2)$
Bob's algorithm: $g(n) \in O(n)$

Simplifying functions of running times

- Ignore the constants.
- Ignore low-order functions.

$$f(n) = n^4 + 3n^3 + n \log n + 500000$$

$$f(n) \in$$

Simplifying functions of running times

- Ignore the constants.
- Ignore low-order functions.

$$f(n) = n^4 + 3n^3 + n\log n + 500000$$
$$f(n) \in$$

Simplifying functions of running times

- Ignore the constants.
- Ignore low-order functions.

$$f(n) = n^4 + 3n^3 + n \log n + 500000$$

$$f(n) \in$$

Simplifying functions of running times

- Ignore the constants.
- Ignore low-order functions.

$$
\begin{aligned}
f(n) &= n^4 + 3n^3 + n\log n + 500000 \\
f(n) &\in O(n^4)
\end{aligned}
$$

Function growth

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n \log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$$

$$n \in O(n)$$

Function growth

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n\log n < n\log^2 n < n\sqrt{n} < n^2 < n^2\log n < n^3 < 2^n < n! < n^n$$

$$\begin{aligned} n &\in O(n) \\ &\in O(n\log n) \end{aligned}$$

Function growth

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n \log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$$

$$\begin{aligned} n &\in O(n) \\ &\in O(n \log n) \\ &\in O(n^4 \log n) \end{aligned}$$

Function growth

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n \log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$$

$$\begin{aligned} n &\in O(n) \\ &\in O(n \log n) \\ &\in O(n^4 \log n) \\ &\in O(n!) \end{aligned}$$

Function growth

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n \log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$$

Example: $f(n) = 5n \log n + n^2 + 23$:

$$\begin{aligned} n &\in O(n) \\ &\in O(n \log n) \\ &\in O(n^4 \log n) \\ &\in O(n!) \end{aligned}$$

Function growth

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n \log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$$

$$\begin{aligned} n &\in O(n) \\ &\in O(n \log n) \\ &\in O(n^4 \log n) \\ &\in O(n!) \end{aligned}$$

Example: $f(n) = 5n \log n + n^2 + 23$:

- $f(n) \in O(n^2)$
- $f(n) \in O(n^2 \log n)$
- What about $O(n \log n)$?

Some examples

$$
\begin{aligned}
f(n) = \quad & 4n^3 \log n + 12\sqrt{n} && \in \\
g(n) = \quad & n\sqrt{n} + n \log n && \in \\
h(n) = \quad & 2n^4 + \sqrt[4]{n+4} && \in
\end{aligned}
$$

Some examples

$$
\begin{aligned}
f(n) &= \quad 4n^3 \log n + 12\sqrt{n} && \in \\
g(n) &= \quad n\sqrt{n} + n \log n && \in \\
h(n) &= \quad 2n^4 + \sqrt[4]{n+4} && \in
\end{aligned}
$$

Some examples

$$
\begin{aligned}
f(n) = &\quad 4n^3 \log n + 12\sqrt{n} && \in O(n^3 \log n) \\
g(n) = &\quad n\sqrt{n} + n \log n && \in \\
h(n) = &\quad 2n^4 + \sqrt[4]{n+4} && \in
\end{aligned}
$$

Some examples

$$
\begin{aligned}
f(n) = &\quad 4n^3 \log n + 12\sqrt{n} &&\in O(n^3 \log n) \\
g(n) = &\quad {\color{red} n\sqrt{n}} + n \log n &&\in \\
h(n) = &\quad 2n^4 + \sqrt[4]{n+4} &&\in
\end{aligned}
$$

Some examples

$$
\begin{aligned}
f(n) = &\quad 4n^3 \log n + 12\sqrt{n} &&\in O(n^3 \log n) \\
g(n) = &\quad n\sqrt{n} + n \log n &&\in O(n\sqrt{n}) \\
h(n) = &\quad 2n^4 + \sqrt[4]{n+4} &&\in
\end{aligned}
$$

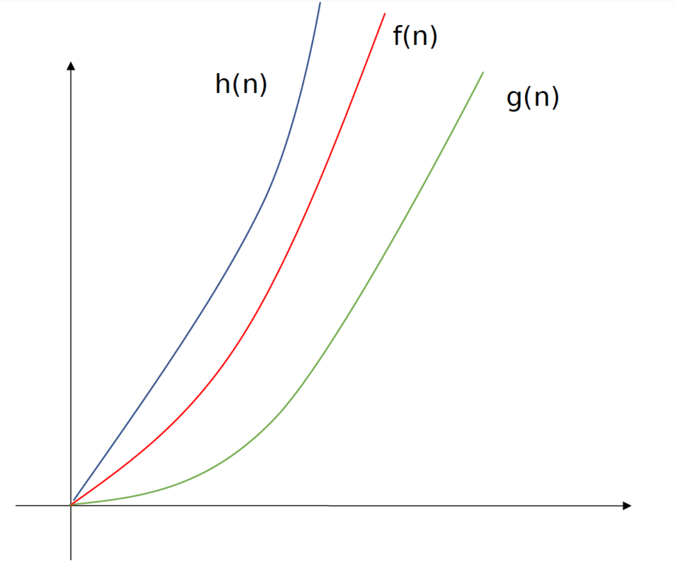
Some examples

$$
\begin{aligned}
f(n) &= 4n^3 \log n + 12\sqrt{n} && \in O(n^3 \log n) \\
g(n) &= n\sqrt{n} + n \log n && \in O(n\sqrt{n}) \\
h(n) &= {\color{red}2n^4} + \sqrt[4]{n+4} && \in
\end{aligned}
$$

Some examples

$$
\begin{aligned}
f(n) &= \quad 4n^3 \log n + 12\sqrt{n} && \in O(n^3 \log n) \\
g(n) &= \quad n\sqrt{n} + n \log n && \in O(n\sqrt{n}) \\
h(n) &= \quad 2n^4 + \sqrt[4]{n+4} && \in O(n^4)
\end{aligned}
$$

Figure of the example expresses



Another example

- Consider the following function.

$$f(n) = \begin{cases} 54n + 9 & , n \text{ is even} \\ n\sqrt{n} + n \log n + 1 & , n \text{ is odd} \end{cases}$$

$$f(n) \in$$

Another example

- Consider the following function.

$$f(n) = \begin{cases} 54n + 9 & , n \text{ is even} \\ n\sqrt{n} + n\log n + 1 & , n \text{ is odd} \end{cases}$$

$$f(n) \in$$

Another example

- Consider the following function.

$$f(n) = \begin{cases} 54n + 9 & , n \text{ is even} \\ n\sqrt{n} + n \log n + 1 & , n \text{ is odd} \end{cases}$$

$$f(n) \in O(n\sqrt{n})$$

Counting steps based on line of code

```
void main(int A[]){
    for(int m=0; m<A.length; m++){
        int i;
        int j;
        j = 0;
        i = 23;
        i = i*j;
    }
}
```

$f(n) \in$

```
void main(int A[]){
    for(int m=0; m<A.length; m++){
        int i = 23;
        int j = 0;
        i = i*j;
    }
}
```

$f(n) \in$

Counting steps based on line of code

```
void main(int A[]){
    for(int m=0; m<A.length; m++){
        int i;
        int j;
        j = 0;
        i = 23;
        i = i*j;
    }
}
```

$$f(n) \in O(n)$$

```
void main(int A[]){
    for(int m=0; m<A.length; m++){
        int i = 23;
        int j = 0;
        i = i*j;
    }
}
```

$$f(n) \in$$

Counting steps based on line of code

```
void main(int A[]){
    for(int m=0; m<A.length; m++){
        int i;
        int j;
        j = 0;
        i = 23;
        i = i*j;
    }
}
```

$$f(n) \in O(n)$$

```
void main(int A[]){
    for(int m=0; m<A.length; m++){
        int i = 23;
        int j = 0;
        i = i*j;
    }
}
```

$$f(n) \in O(n)$$

Effect of the loop in the running time

for $i = 0 \to n - 1$ **do**
  $i \leftarrow i + 2;$

for $i = 0 \to 20$ **do**
  $i \leftarrow i \times 2;$

Effect of the loop in the running time

for $i = 0 \to n-1$ **do**
  $i \leftarrow i + 2;$

for $i = 0 \to 20$ **do**
  $i \leftarrow i \times 2;$

- $f(n) = \sum_{i=0}^{n-1} 1 = n$

- $g(n) = \sum_{i=0}^{19} 1 = 20$

Effect of the loop in the running time

for $i = 0 \to n - 1$ **do**
  $i \leftarrow i + 2;$

- $f(n) = \sum_{i=0}^{n-1} 1 = n$
- $f(n) \in O(n)$

for $i = 0 \to 20$ **do**
  $i \leftarrow i \times 2;$

- $g(n) = \sum_{i=0}^{19} 1 = 20$
- $g(n) \in O(1)$

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0$;
$i \leftarrow 0$;
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1$;
  $result \leftarrow \textsc{Add}(i, result)$;

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0$;
$i \leftarrow 0$;
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1$;
  $result \leftarrow \textsc{Add}(i, result)$;

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0$;
$i \leftarrow 0$;
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1$;
  $result \leftarrow \textsc{Add}(i, result)$;

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y
return $x + y$

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0;$
$i \leftarrow 0;$
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1;$
  $result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y
return $x + y$

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0;$
$i \leftarrow 0;$
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1;$
  $result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y
return $x + y$

$$f(n) = 2 + \sum_{i=1}^{n-1} (1 + h(n))$$

Effect of the loop in the running time

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0$;
$i \leftarrow 0$;
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1$;
  $result \leftarrow \textsc{Add}(i, result)$;

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y
return $x + y$

$$f(n) = 2 + \sum_{i=1}^{n-1}(1 + h(n))$$

$$f(n) \in O(n)$$

Big-Oh

- Upper bound
- Highest order term in the function identifies it
- Asymptotic notation gives more detail
  - $n \in O(n)$
  - $n \in O(n^4 \log n)$
  - $n \in O(n!)$

Does $f(n) = O(g(n))$ holds?

- We should check whether $f(n) \leq g(n)$.
- We only consider the highest order function.
- We ignore constants.

Formal definition of big-Oh

$f(n)$ is $O(g(n))$ if there exist a constant $M > 0$ and $n_0 > 0$ such that:

$$f(n) \leq M \times g(n), \text{ for } n > n_0$$

We ignore the constants and consider the highest order term.

An example

$f(n) = n^2 + 13 > 50$ for large values of n:

$$n = 6 \Rightarrow \qquad f(6) = 77 > 50$$

$$n = 7 \Rightarrow \qquad f(7) = 82 > 50$$

Formal definition of big-Oh

$f(n)$ is $O(g(n))$ if there exist a constant $M > 0$ and $n_0 > 0$ such that:

$$f(n) \leq M \times g(n), \text{ for } n > n_0$$

We ignore the constants, consider the highest order term, **and** the following holds for the quantifiers $\exists$ and $\forall$:

$$f(n) \in O(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)$$

Universal quantifier ($\forall$) and existential quantifier ($\exists$)

- For all

$$\forall x > 2, x^3 + 2 > 10$$

- For some / There exists

$$\exists x > 0, \text{s.t. } x^2 = 64$$
$$\exists x, \text{s.t. } x^2 = 64$$

Quantifiers in Calculus

$$\lim_{x \to \infty} f(x) = c$$

$$\forall \epsilon \exists \delta \text{ s.t. } |x - a| < \delta \to |f(x) - c| < \epsilon$$

Formal definition of big-Oh

$f(n)$ is $O(g(n))$ if there exist a constant $M > 0$ and $n_0 > 0$ such that:

$$f(n) \leq M \times g(n), \text{ for } n > n_0$$

We ignore the constants, consider the highest order term, **and** the following holds for the quantifiers $\exists$ and $\forall$:

$$f(n) \in O(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)$$

Set notation

$$f(n) \in O(g(n)) \text{ or } f(n) = O(g(n))$$

This does not mean that $f(n) = g(n)$.

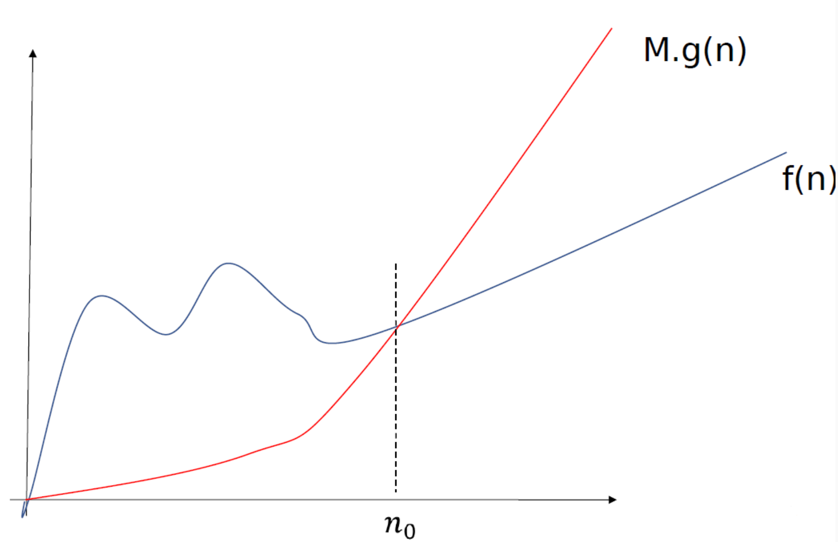
Set notation

$$f(n) \in O(g(n)) \text{ or } f(n) = O(g(n))$$

This does not mean that $f(n) = g(n)$.

$$n = O(n^2)$$
$$n^2 = O(n^2)$$
$$n \neq n^2$$

Formal definition of $f(n) \in O(g(n))$ expresses



Is $f(n) \in O(g(n))$?

We need to show that the following holds.

$$f(n) \in O(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)$$

An example

- Does $(n^3 + 4) \in O(n^3)$ hold?
- For $M = 2$ and $n_0 = 3$:

$$
\begin{aligned}
n &\geq 3 \\
n^3 &\geq 27 \\
n^3 &\geq 4 \\
2n^3 &\geq 4 + n^3 \\
M.g(n) &\geq f(n)
\end{aligned}
$$

Negating quantifiers

- “Not every snowy day is cloudy.” $=$ “There are some snowy days that are sunny.”
- “Not every animal is a dog.” $=$ “There are some animals that are dogs.”
- “Not every person drives a car.” $=$ “There are some people who drive.”

$$\neg\left[\forall x P(x)\right] = \exists x \neg P(x)$$
$$\neg\left[\exists x P(x)\right] = \forall x \neg P(x)$$

Is $f(n) \notin O(g(n))$?

- We need to show that the negation holds.

$$\neg[f(n) \in O(g(n))]$$

$$\neg[\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)]$$

$$\forall M > 0, \forall n_0 > 0 \text{ s.t. } \exists n > n_0, f(n) > M.g(n)$$

An example

$$(2n^2 + 3) \notin O(n)$$

$$\forall M > 0, \forall n_0 > 0 \text{ s.t. } \exists n \geq n_0, 2n^2 + 3 > M.n$$

$$n = M + n_0 + 1$$

$$n > 1$$

$$n^2 > n$$

$$n^2 + 3 > n$$

$$n > M$$

$$n^2 > M.n$$

$$n^2 > (M - 1).n$$

$$2n^2 + 3 > M.n$$

Diagram of proof

$$
\begin{tikzcd}
 & & & n = M + n_0 + 1 \arrow[dl] \arrow[dr] & & & \\
 & & n > 1 \arrow[dl] & & n > M \arrow[dr] & & \\
 & n^2 > n \arrow[dl] & & & & n^2 > M.n \arrow[dr] & \\
n^2 + 3 > n \arrow[drrr] & & & & & & n^2 > (M-1).n \arrow[dlll] \\
 & & & 2n^2 + 3 > M.n & & &
\end{tikzcd}
$$

CPS 616: Algorithms

Asymptotic Notation - Part II

January 25, 2022

Onur Çağırıcı

How to do the proofs?

- An example: $(2n^2 + 50) \in O(n^2)$
  - Work backwards.
  - To prove $f(n) \in O(g(n))$, we first find appropriate values of M and n_0.

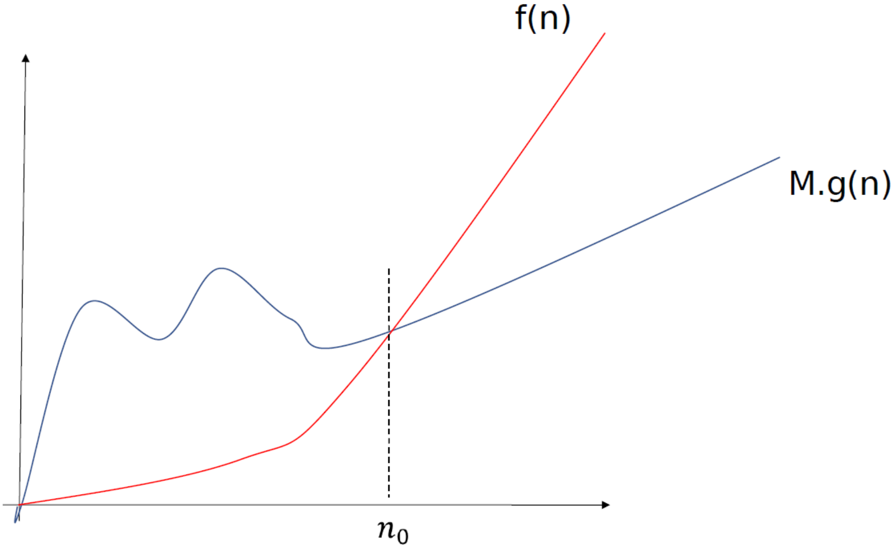
Formal definition of Ω = the asymptotic lower bound

$f(n)$ is $\Omega(g(n))$ if there exist two constants $M > 0$ and $n_0 > 0$ such that:

$$f(n) \geq M.g(n), \text{ for } n > n_0$$

$$f(n) \in \Omega(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \geq M.g(n)$$

Formal definition of $f(n) \in \Omega(g(n))$ expresses



An example

- $f(n) = 8n \log n$
- $g(n) = 4n$
- Does $f(n) \in \Omega(g(n))$ hold?

$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \geq M.g(n)$$

Example cont'd

- Does $8n \log n \in \Omega(4n)$ hold?
- For $M = 2$ and $n_0 = 10$:

$$
\begin{aligned}
n &> 10 \\
\log n &> \log 10 \\
\log n &> 1 \\
n \log n &> n \\
8n \log n &> 8n \\
8n \log n &> 2 \times 4n \\
f(n) &> M.g(n) \\
f(n) &\geq M.g(n)
\end{aligned}
$$

Omega is a lower bound

$1 < logn < n^{1/4} < \sqrt{n} < n < nlogn < nlog^2n < n\sqrt{n} < n^2 < n^2logn < n^3 < 2^n < n! < n^n$

$$10n^4 \in \Omega(n^4)$$
$$10n^4 \in \Omega(n^3)$$
$$10n^4 \in \Omega(n^2logn)$$
$$10n^4 \notin \Omega(n^5)$$

Theorem

- Given any two functions $f : R^+ \to R^+$ and $g : R^+ \to R^+$, the following holds:

$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)).$$

Proof:

Theorem

- Given any two functions $f : R^+ \to R^+$ and $g : R^+ \to R^+$, the following holds:
$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)).$$

Proof:

$(\Rightarrow)$ $f(n) \in O(g(n))$

Theorem

- Given any two functions $f : R^+ \rightarrow R^+$ and $g : R^+ \rightarrow R^+$, the following holds:
$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)).$$

Proof:

$(\Rightarrow)$ $f(n) \in O(g(n))$

$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)$$
$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, M.g(n) \geq f(n)$$
$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, g(n) \geq \frac{1}{M} f(n)$$
$$\exists M' > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, g(n) \geq M' f(n) \text{ where } M' = \frac{1}{M}$$

Theorem

- Given any two functions $f : R^+ \to R^+$ and $g : R^+ \to R^+$, the following holds:
$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)).$$

Proof:

$(\Rightarrow)$ $f(n) \in O(g(n))$

$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)$$
$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, M.g(n) \geq f(n)$$
$$\exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, g(n) \geq \frac{1}{M} f(n)$$
$$\exists M' > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, g(n) \geq M' f(n) \text{ where } M' = \frac{1}{M}$$

Theta notation

- $f(n) = n^2 \log n + n \log n + 3n + 12$

$$f(n) \in O(n^4)$$
$$f(n) \in O(n^2 \log n)$$

- $O(n^2 \log n)$ expresses $f(n)$ better.

Theta notation

$$f(n) = n^2 \log n + n \log n + 3n + 12$$

Theta notation

$$f(n) = n^2 \log n + n \log n + 3n + 12$$
$$f(n) \in O(n^2 \log n)$$

Theta notation

$$f(n) = n^2 \log n + n \log n + 3n + 12$$
$$f(n) \in O(n^2 \log n)$$
$$f(n) \in \Omega(n^2 \log n)$$

Theta notation

$$f(n) = n^2 \log n + n \log n + 3n + 12$$

$$f(n) \in O(n^2 \log n)$$

$$f(n) \in \Omega(n^2 \log n)$$

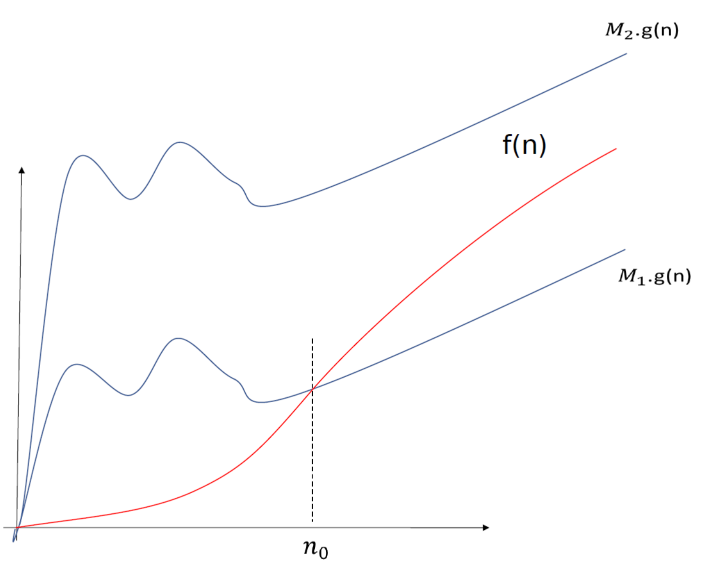
$$f(n) \in \Theta(n^2 \log n)$$

Formal definition of Θ = the asymptotic average bound

$f(n)$ is $\Theta(g(n))$ if there exist three constants $M_1 > 0$, $M_2 > 0$ and $n_0 > 0$ such that;

$$M_1.g(n) \leq f(n) \leq M_2.g(n), \text{ for } n > n_0$$

Formal definition of $f(n) \in \Theta(g(n))$ expresses



Theta is an asymptotic tight bound (upper and lower bound)

$$23n^3 \log n + n^2 + 13n \in \Theta(n^3 \log n)$$

$$23n^3 \log n + n^2 + 13n \in O(n^3 \log n)$$

$$23n^3 \log n + n^2 + 13n \in O(n^4)$$

$$23n^3 \log n + n^2 + 13n \in \Omega(n^3 \log n)$$

$$23n^3 \log n + n^2 + 13n \in \Omega(n^3)$$

$$23n^3 \log n + n^2 + 13n \notin \Theta(n^3)$$

$$23n^3 \log n + n^2 + 13n \notin \Theta(n^4)$$

An example

- $f(n) = 3n^3 + 9$
- $g(n) = n^3$
- Does $f(n) \in \Theta(g(n))$ hold?
- For $M_1 = 3$, $M_2 = 4$ and $n_0 = 3$:

$$n > 3$$

$$n^3 > 27$$

$$n^3 > 9$$

$$3n^3 + 9 < 3n^3 + n^3 = 4n^3$$

$$3n^3 < 3n^3 + 9$$

$$M_1.g(n) \leq f(n) \leq M_2.g(n)$$

Theorem

- Given any two functions $f : R^+ \to R^+$ and $g : R^+ \to R^+$, the following holds:

$$f(n) \in \Theta(g(n)) \iff [f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(g(n))].$$

Proof: From the definitions of O, Ω and Θ.

Little-Oh notation

- $f(n) \in O(n^3)$
- $f(n) \notin \Theta(n^3)$
- An example:

$$n^{1-\epsilon} \in O(n), \text{ but } n^{1-\epsilon} \notin \Theta(n)$$
$$n^{2-\epsilon} \in O(n^2), \text{ but } n^{2-\epsilon} \notin \Theta(n^2)$$

- Thus, the following hold:

$$n^{1-\epsilon} \in o(n)$$
$$n^{2-\epsilon} \in o(n^2)$$

Formal definition of Little-Oh

$f(n)$ is $o(g(n))$ if $g(n)$ is an upper bound for $f(n)$ **but** they grow with different rates.

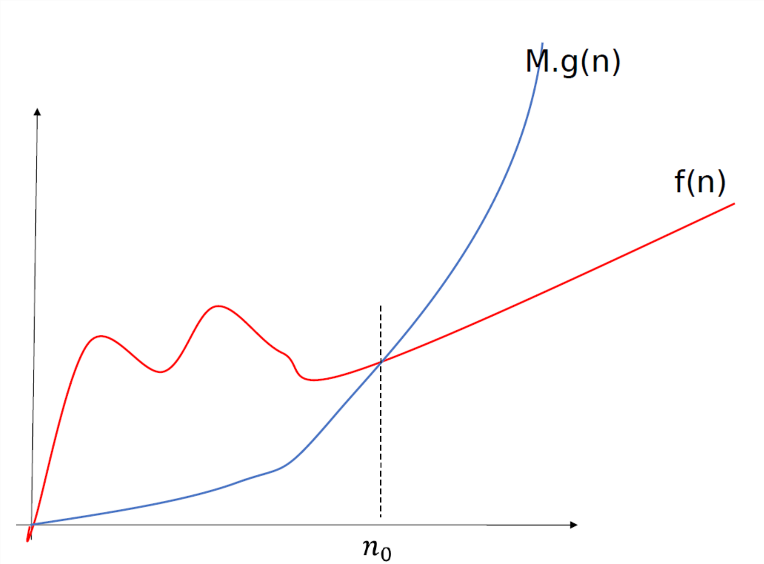
Formal definition of Little-Oh

$f(n)$ is $o(g(n))$ if $g(n)$ is an upper bound for $f(n)$ **but** they grow with different rates.

Formally:

$$f(n) \in o(g(n)) \iff \forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n).$$

Formal definition of $f(n) \in o(g(n))$ expresses



An example

- $f(n) = 2n + 10$
- $g(n) = n^2$
- Does $f(n) \in o(g(n))$ hold?

$$
\begin{tikzcd}
 & & & \forall M, n_0 = max\{\sqrt{\frac{20}{M}}, \frac{4}{M}\} + 1 \arrow[dl] \arrow[dr] & & & \\
 & & n > \sqrt{\frac{20}{M}} \arrow[dl] & & n > \frac{4}{M} \arrow[dr] & & \\
 & n^2 > \frac{20}{M} \arrow[dl] & & & & \frac{M.n}{2} > 2 \arrow[dr] & \\
\frac{M.n^2}{2} > 10 \arrow[drrr] & & & & & & \frac{M.n^2}{2} > 2n \arrow[dlll] \\
 & & & M.n^2 > 2n + 10 & & &
\end{tikzcd}
$$

- Therefore, $M.g(n) \geq f(n)$ holds.

Is $f(n) \notin o(g(n))$?

We need to show that the negation holds.

$$\neg[\forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M.g(n)]$$
$$\exists M > 0, \forall n_0 > 0 \text{ s.t. } \exists n > n_0, f(n) > M.g(n)$$

Little omega notation

- $f(n) \in \Omega(g(n))$
- $f(n) \notin \Theta(g(n))$
- An example:

$$n^{1+\epsilon} \in \Omega(n), \text{ but } n^{1+\epsilon} \notin \Theta(n)$$

$$n^{2+\epsilon} \in \Omega(n^2), \text{ but } n^{2+\epsilon} \notin \Theta(n^2)$$

- Thus, the followings hold:

$$n^{1+\epsilon} \in \omega(n)$$

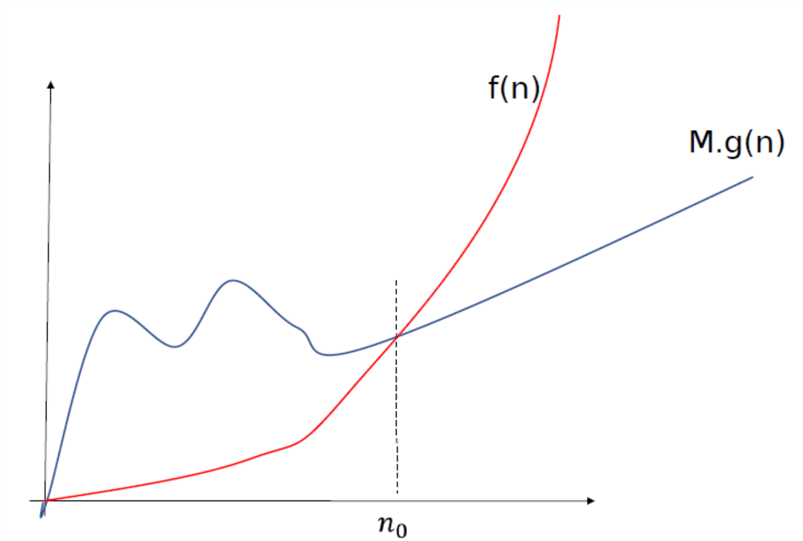
$$n^{2+\epsilon} \in \omega(n^2)$$

Formal definition of Little omega

$g(n)$ is a lower bound for $f(n)$ but they grow with different rates. Formally:

$$f(n) \in \omega(g(n)) \iff \forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \geq M.g(n).$$

Formal definition of $f(n) \in \omega(g(n))$ expresses



An example

- $f(n) = 16n \log n + 4 \log n + 63$

$$f(n) \in O(n \log n)$$
$$f(n) \in \Omega(n \log n)$$
$$f(n) \in \Theta(n \log n)$$
$$f(n) \notin o(n \log n)$$
$$f(n) \notin \omega(n \log n)$$

Example cont'd

- $f(n) = 16n \log n + 4 \log n + 63$

$$f(n) \in O(n^2 \log n)$$
$$f(n) \notin \Omega(n^2 \log n)$$
$$f(n) \notin \Theta(n^2 \log n)$$
$$f(n) \in o(n^2 \log n)$$
$$f(n) \notin \omega(n^2 \log n)$$

Example cont'd

- $f(n) = 16n \log n + 4 \log n + 63$

$$f(n) \notin O(\log n)$$
$$f(n) \in \Omega(\log n)$$
$$f(n) \notin \Theta(\log n)$$
$$f(n) \notin o(\log n)$$
$$f(n) \in \omega(\log n)$$

Theorem

- Given any two functions $f : R^+ \to R^+$ and $g : R^+ \to R^+$, the following holds:

$$f(n) \in o(g(n)) \iff g(n) \in \omega(f(n)).$$

- Proof $(\to)$ $f(n) \in o(g(n))$

$$\forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \le M.g(n)$$

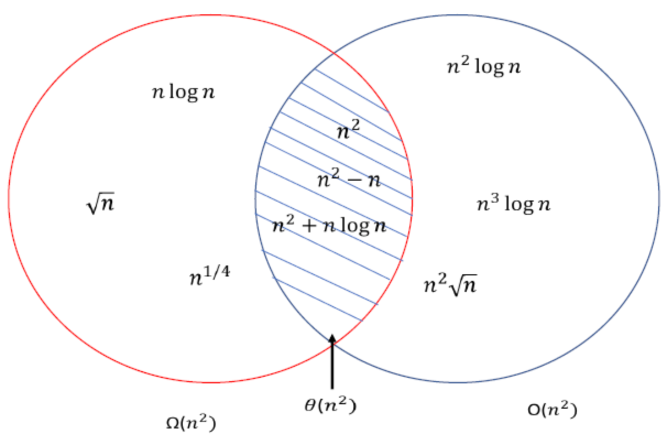
$$\forall M > 0, \exists n_0 \text{ s.t. } \forall n > n_0, M.g(n) \ge f(n)$$

$$\forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, g(n) \ge \frac{1}{M} f(n)$$

$$\forall M' > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, g(n) \ge M'.f(n) \text{ where } M' = \frac{1}{M}$$

- $(\leftarrow)$ $g(n) \in \omega(f(n))$ proved analogously.

Overview



$n \log n$
$\sqrt{n}$
$n^{1/4}$
n^2
$n^2 - n$
$n^2 + n \log n$
$n^2 \log n$
$n^3 \log n$
$n^2\sqrt{n}$
$\Omega(n^2)$
$\theta(n^2)$
$O(n^2)$

What in mathematics gives us the growth of functions?

- $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$
- For some nice function, limit gives us the answer.

$$c = 0 \to f(n) \in o(g(n))$$
$$c = \infty \to f(n) \in \omega(g(n))$$
$$c \in R^{+} \to f(n) \in \Theta(g(n))$$

Some examples

- $\lim_{n\to\infty} \frac{12n^3+23n}{6n^3}$
- $\lim_{n\to\infty} \frac{2n^6+12}{6n^5}$
- $\lim_{n\to\infty} \frac{2n^2+1}{6n^3}$

An important reminder

- For some functions, the limit does not exist.

An important reminder

- For some functions, the limit does not exist.
- Consider two functions

An important reminder

- For some functions, the limit does not exist.
- Consider two functions
  - $f(n) = 3\cos n$

An important reminder

- For some functions, the limit does not exist.
- Consider two functions
  - $f(n) = 3\cos n$
  - $g(n) = 12$.

An important reminder

- For some functions, the limit does not exist.
- Consider two functions
  - $f(n) = 3\cos n$
  - $g(n) = 12$.
- Instead of $\lim_{n\to\infty} \frac{3\cos n}{12}$, we must use the definition.

Summary

- To determine the cost of a given algorithm:

Summary

- To determine the cost of a given algorithm:
    1. Find the cost function

Summary

- To determine the cost of a given algorithm:
  1. Find the cost function
  2. Express it by asymptotic notations

Summary

- To determine the cost of a given algorithm:
    1. Find the cost function
    2. Express it by asymptotic notations
- Asymptotic notations are the tools to help us compare the complexity of algorithms.

CPS 616: Algorithms

Recurrence Relations - Part I

February 2, 2022

Onur Çağırıcı

References

- Chapters
  - 4.2
  - 4.3
  - 4.4
  - 4.5

Cost function

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers
Output: Minimum integer in A

Cost function

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers
Output: Minimum integer in A

```
min ← A[0];
i ← 0;
while i < n do
    if A[i] < min then
        min ← A[i];
    i ← i + 1;
return min
```

Cost function

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers
Output: Minimum integer in A

$min \leftarrow A[0]$;
$i \leftarrow 0$;
while $i < n$ **do**
  if $A[i] < min$ **then**
    $min \leftarrow A[i]$;
  $i \leftarrow i + 1$;
return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$

Cost function

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers
Output: Minimum integer in A

$min \leftarrow A[0]$;
$i \leftarrow 0$;
while $i < n$ **do**
  if $A[i] < min$ **then**
    $min \leftarrow A[i]$;
  $i \leftarrow i + 1$;
return min

$$f(n) = 1 + \sum_{i=0}^{n-1} 3$$
$$= 1 + 3n$$

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of all integers from 1 to n

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0$;
$i \leftarrow 0$;
for $j \leftarrow 1 \rightarrow n$ **do**
    $i \leftarrow i + 1$;
    $result \leftarrow \textsc{Add}(i, result)$;

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0;$
$i \leftarrow 0;$
for $j \leftarrow 1 \rightarrow n$ **do**
    $i \leftarrow i + 1;$
    $result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of all integers from 1 to n

$result \leftarrow 0;$

$i \leftarrow 0;$

for $j \leftarrow 1 \rightarrow n$ **do**

  $i \leftarrow i + 1;$

  $result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y

Output: Sum of x and y

return $x + y$

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0$;
$i \leftarrow 0$;
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1$;
  $result \leftarrow \textsc{Add}(i, result)$;

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y
return $x + y$

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of all integers from 1 to n

$result \leftarrow 0;$
$i \leftarrow 0;$
for $j \leftarrow 1 \rightarrow n$ **do**
  $i \leftarrow i + 1;$
  $result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y
Output: Sum of x and y
return $x + y$

$$f(n) = 2 + \sum_{i=1}^{n-1}(1 + h(n))$$

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of all integers from 1 to n

$result \leftarrow 0;$

$i \leftarrow 0;$

for $j \leftarrow 1 \rightarrow n$ **do**

  $i \leftarrow i + 1;$

  $result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y

Output: Sum of x and y

return $x + y$

$$f(n) = 2 + \sum_{i=1}^{n-1}(1 + h(n))$$

$$f(n) \in O(n)$$

Let's solve: addition of n consecutive integers

for $j = 1 \to n$ **do**

$\quad i = i + 1;$

$\quad result \leftarrow \textsc{Add}(i, result);$

Let's solve: addition of n consecutive integers

for $j = 1 \to n$ **do**

$\quad i = i + 1;$

$\quad result \leftarrow \textsc{Add}(i, result);$

$$\underbrace{1 + 2 + 3 + 4 + 5 + \cdots + (n-2) + (n-1)}_{\textsc{Add}(n-1)} + n$$

Let's solve: addition of n consecutive integers

for $j = 1 \to n$ **do**

  $i = i + 1;$

  $result \leftarrow \textsc{Add}(i, result);$

$$\underbrace{1 + 2 + 3 + 4 + 5 + \cdots + (n-2) + (n-1)}_{\textsc{Add}(n-1)} + n$$

$$\textsc{Add}(n) \begin{cases} 1 & , n = 1 \\ \textsc{Add}(n-1) + n & , n \neq 1 \end{cases}$$

Let's solve: addition of n consecutive integers

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of integers from 1 to n

if $n = 1$ **then return** 1;

else return $\textsc{Sum}(n+1) + n$;

Let's solve: addition of n consecutive integers

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of integers from 1 to n

if $n = 1$ **then return** 1;

else return $\textsc{Sum}(n+1) + n$;

$$f(n) \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Let's solve: addition of n consecutive integers

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of integers from 1 to n
if $n = 1$ **then return** 1;
else return $\textsc{Sum}(n+1) + n$;

$$f(n) \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Let's solve: factorial

Algorithm: $\textsc{Fac}(n)$

Input: An integer n

Output: Product of integers from 1 to n

if $n = 1$ **then return** 1;

else return $\textsc{Fac}(n-1) \times n$;

Let's solve: factorial

Algorithm: $\textsc{Fac}(n)$

Input: An integer n
Output: Product of integers from 1 to n
if $n = 1$ **then return** 1;
else return $\textsc{Fac}(n-1) \times n$;

$$f(n) \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Let's solve: factorial

Algorithm: $\textsc{Fac}(n)$

Input: An integer n
Output: Product of integers from 1 to n
if $n = 1$ **then return** 1;
else return $\textsc{Fac}(n-1) \times n$;

$$f(n) \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Solving recurrence relations

- We need to have the time $f(n)$ in terms of n, NOT the function f!
- Thus, we need to solve recurrence relations.

Solving recurrence relations

- Find a recurrence relation
- Solve the recurrence relation
- Correctness proof by induction

$$\textsc{Add}(n) \begin{cases} 1 & , n = 1 \\ \textsc{Add}(n-1) + n & , n \neq 1 \end{cases}$$

Solving recurrence relations

- Find a recurrence relation
- Solve the recurrence relation
- Correctness proof by induction

$$\textsc{Add}(n) \begin{cases} 1 & , n = 1 \\ \textsc{Add}(n-1) + n & , n \neq 1 \end{cases}$$

$$f(n) = O(n)$$

Solving recurrence relations

- Substitution method
- Recurrence tree
- Master Theorem

Solving recurrence relations – Substitution method

Assume $f(1) = 1$

$$
\begin{aligned}
f(n) &= f(n-1) + 1 \\
&= f(n-2) + 1 + 1 \\
&= f(n-3) + 1 + 1 + 1 \\
&\vdots \\
&= f(1) + \underbrace{1 + 1 + \cdots + 1}_{n-1} \\
&= \overbrace{1}^{f(1)} + n - 1
\end{aligned}
$$

Induction

To prove $\forall k \geq P(k)$

1: Show that base case $P(1)$ holds.
2: Show that $\forall k \geq 1,\ P(k) \implies P(k+1)$

Strong Induction

To prove $\forall k \geq P(k)$

1: Show that base case $P(1)$ holds.
2: Show that $\forall k \geq 1$,
$(P(1) \wedge P(2) \wedge \cdots \wedge P(k)) \implies P(k+1)$

Solving recurrence relations – Substitution method

- **Base case:** $P(c)$ holds for some $c \in \mathbb{Z}$,
- **Hypothesis:** Assume that $P(k)$ holds for some $k \geq c$,
- **Induction:** $P(k+1)$ holds.
- **Conclusion:** $\forall n \geq c,\ P(n)$ holds.

Let's solve: addition of n consecutive integers

Show $\forall n \geq 1,\ f(n) = n$

- **Base case:** $f(1) = 1$
- **Hypothesis:** for some $k \geq 1,\ f(k) = k$
- **Inductive step:** $f(k+1) = f(k) + 1$
  - By the recursion: $f(k+1) = f(k) + 1$
  - By hypothesis: $f(k) = k \Rightarrow f(k+1) = k + 1$
- **Conclusion:** The base holds, and the inductive hypothesis concludes the inductive step. Therefore, $f(n) = n \quad \square$

Let's solve: converting decimal to binary

Algorithm: $\textsc{DecToBin}(n)$

Input: A positive integer n

Output: Binary representation of n, **printed on screen**

Let's solve: converting decimal to binary

Algorithm: $\textsc{DecToBin}(n)$

Input: A positive integer n
Output: Binary representation of n, **printed on screen**

if $n = 1$ **then** $\textsc{Print}(n)$;
else
  $\textsc{DecToBin}(n/2)$;
  $\textsc{Print}(n \mod 2)$;

Let's solve: converting decimal to binary

Algorithm: $\textsc{DecToBin}(n)$

Input: A positive integer n

Output: Binary representation of n, **printed on screen**

if $n = 1$ **then** $\textsc{Print}(n)$;

else

  $\textsc{DecToBin}(n/2)$;

  $\textsc{Print}(n \mod 2)$;

Recurrence relation

Let's solve: converting decimal to binary

Algorithm: $\textsc{DecToBin}(n)$

Input: A positive integer n

Output: Binary representation of n, **printed on screen**

if $n = 1$ **then** $\textsc{Print}(n)$;

else

  $\textsc{DecToBin}(n/2)$;

  $\textsc{Print}(n \mod 2)$;

Recurrence relation

$$T(n) \begin{cases} 1 & , n = 1 \\ T\left(\frac{n}{2}\right) + 1 & , \text{otherwise} \end{cases}$$

Let's solve: converting decimal to binary

Algorithm: $\textsc{DecToBin}(n)$

Input: A positive integer n
Output: Binary representation of n, **printed on screen**

if $n = 1$ **then** $\textsc{Print}(n)$;
else
  $\textsc{DecToBin}(n/2)$;
  $\textsc{Print}(n \mod 2)$;

Recurrence relation

$$T(n) \begin{cases} 1 & , n = 1 \\ T\left(\frac{n}{2}\right) + 1 & \text{, otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= \left(\frac{n}{2}\right) + 1 \\ &= \left(\frac{n}{2^2}\right) + 1 + 1 \\ &= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\ &\vdots \end{aligned}$$

Let's solve: converting decimal to binary

$$
\begin{aligned}
T(n) &= \left(\frac{n}{2}\right) + 1 \\
&= \left(\frac{n}{2^2}\right) + 1 + 1 \\
&= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\
&\vdots
\end{aligned}
$$

Let's solve: converting decimal to binary

$$
\begin{aligned}
T(n) &= \left(\frac{n}{2}\right) + 1 \\
&= \left(\frac{n}{2^2}\right) + 1 + 1 \\
&= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\
&\vdots \\
&= \left(\frac{n}{2^i}\right) + \underbrace{1 + \cdots + 1}_{i}
\end{aligned}
$$

Let's solve: converting decimal to binary

$$
\begin{aligned}
T(n) &= \left(\frac{n}{2}\right) + 1 \\
&= \left(\frac{n}{2^2}\right) + 1 + 1 \\
&= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\
&\vdots \\
&= \left(\frac{n}{2^i}\right) + \underbrace{1 + \cdots + 1}_{i} \\
&= T(1) + \log_2(n) \\
&= \log_2(n) + 1
\end{aligned}
$$

Let's solve: converting decimal to binary

$$\begin{aligned}
T(n) &= \left(\frac{n}{2}\right) + 1 \\
&= \left(\frac{n}{2^2}\right) + 1 + 1 \\
&= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\
&\vdots \\
&= \left(\frac{n}{2^i}\right) + \underbrace{1 + \cdots + 1}_{i} \\
&= T(1) + \log_2(n) \\
&= \log_2(n) + 1
\end{aligned}$$

- **Base case:**
- $i =$

Let's solve: converting decimal to binary

$$
\begin{aligned}
T(n) &= \left(\frac{n}{2}\right) + 1 \\
&= \left(\frac{n}{2^2}\right) + 1 + 1 \\
&= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\
&\vdots \\
&= \left(\frac{n}{2^i}\right) + \underbrace{1 + \cdots + 1}_{i} \\
&= T(1) + \log_2(n) \\
&= \log_2(n) + 1
\end{aligned}
$$

- **Base case:** $\frac{n}{2^i} = 1$
- $i =$

Let's solve: converting decimal to binary

$$
\begin{aligned}
T(n) &= \left(\frac{n}{2}\right) + 1 \\
&= \left(\frac{n}{2^2}\right) + 1 + 1 \\
&= \left(\frac{n}{2^3}\right) + 1 + 1 + 1 \\
&\vdots \\
&= \left(\frac{n}{2^i}\right) + \underbrace{1 + \cdots + 1}_{i} \\
&= T(1) + \log_2(n) \\
&= \log_2(n) + 1
\end{aligned}
$$

- **Base case:** $\frac{n}{2^i} = 1$
- $i = \log_2(n)$

Let's solve: converting decimal to binary

induction

Show $\forall n \geq 1,\ f(n) = 1 + \log_2(n)$

- **Base case**: $f(1) = 1 + \log_2(1)$

Let's solve: converting decimal to binary

induction

Show $\forall n \geq 1,\ f(n) = 1 + \log_2(n)$

- **Base case**: $f(1) = 1 + \log_2(1)$
- **Hypothesis**: For some $k > 1$, for $i \in \{1, 2, \ldots, k\},\ f(k) = 1 + \log_2(k)$

Let’s solve: converting decimal to binary

induction

Show $\forall n \geq 1,\ f(n) = 1 + \log_2(n)$

- **Base case**: $f(1) = 1 + \log_2(1)$
- **Hypothesis**: For some $k > 1$, for $i \in \{1, 2, \ldots, k\},\ f(k) = 1 + \log_2(k)$
- **Inductive step:** $f(k+1) = 1 + \log_2(k+1)$

Let's solve: converting decimal to binary

induction

Show $\forall n \geq 1,\ f(n) = 1 + \log_2(n)$

- **Base case**: $f(1) = 1 + \log_2(1)$
- **Hypothesis**: For some $k > 1$, for $i \in \{1, 2, \ldots, k\},\ f(k) = 1 + \log_2(k)$
- **Inductive step:** $f(k+1) = 1 + \log_2(k+1)$
  - By the recursion: $f(k+1) = f(\frac{k+1}{2}) + 1$

Let's solve: converting decimal to binary

induction

Show $\forall n \geq 1, \ f(n) = 1 + \log_2(n)$

- **Base case**: $f(1) = 1 + \log_2(1)$
- **Hypothesis**: For some $k > 1$, for $i \in \{1, 2, \ldots, k\}, \ f(k) = 1 + \log_2(k)$
- **Inductive step:** $f(k+1) = 1 + \log_2(k+1)$
  - By the recursion: $f(k+1) = f(\frac{k+1}{2}) + 1$
  - By the hypothesis: $f(\frac{k+1}{2}) = 1 + \log_2\left(\frac{k+1}{2}\right)$

Let's solve: converting decimal to binary

How to continue?

$$f(k+1) = \log_2\left(\frac{k+1}{2}\right) + 1 + 1$$

Let's solve: converting decimal to binary

How to continue?

$$
\begin{aligned}
f(k+1) &= \log_2\left(\frac{k+1}{2}\right) + 1 + 1 \\
&= \log_2\left(\frac{k+1}{2}\right) + 2\log_2(2) \\
&= \log_2\left(\frac{k+1}{2}\right) + \log_2(2^2) \\
&= \log_2\left(\frac{k+1}{2} \times 2^2\right) \\
&= \log_2(2(k+1)) \\
&= \cancelto{1}{\log_2(2)} + \log_2(k+1)
\end{aligned}
$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$T(n) \begin{cases} 1 & , n = 1 \\ 3T\left(\frac{n}{4} + 1\right) & , \text{otherwise} \end{cases}$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$T(n) \begin{cases} 1 & , n = 1 \\ 3T\left(\frac{n}{4} + 1\right) & \text{, otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{4}\right) + 1 \\ &= 3\left(3T\left(\frac{n}{4^2}\right) + 1\right) + 1 \\ &= 3^2 T\left(\frac{n}{4^2}\right) + 3 + 1 \end{aligned}$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$T(n) \begin{cases} 1 & , n = 1 \\ 3T\left(\frac{n}{4} + 1\right) & \text{, otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{4}\right) + 1 \\ &= 3\left(3T\left(\frac{n}{4^2}\right) + 1\right) + 1 \\ &= 3^2T\left(\frac{n}{4^2}\right) + 3 + 1 \\ &= 3^2\left(3T\left(\frac{n}{4^3} + 1\right) + 3 + 1\right) \\ &= 3^3T\left(\frac{n}{4^3}\right) + 3^2 + 3 + 1 \\ &\vdots \end{aligned}$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$T(n) \begin{cases} 1 & , n = 1 \\ 3T\left(\frac{n}{4} + 1\right) & \text{, otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{4}\right) + 1 \\ &= 3\left(3T\left(\frac{n}{4^2}\right) + 1\right) + 1 \\ &= 3^2T\left(\frac{n}{4^2}\right) + 3 + 1 \\ &= 3^2\left(3T\left(\frac{n}{4^3} + 1\right) + 3 + 1\right) \\ &= 3^3T\left(\frac{n}{4^3}\right) + 3^2 + 3 + 1 \\ &\vdots \\ &= 3^iT\left(\frac{n}{4^i}\right) + 3^{i-1} + 3^{i-2} + \cdots + 1 \end{aligned}$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$$

- **Base case:**
- $i =$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$$

- **Base case:** $n = 4^i$
- $i =$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$3^i T(\tfrac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$$

- **Base case:** $n = 4^i$
- $i = \log_4(n)$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$

- **Base case:** $n = 4^i$
- $i = \log_4(n)$

Remember

- $S_n = a_0 \dfrac{1 - q^n}{1 - q}$
- $a^{\log_c(b)} = b^{\log_c(a)}$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$

- **Base case:** $n = 4^i$
- $i = \log_4(n)$

$$T(n) = 3^{\log_4(n)} + \sum_{a=0}^{\log_4(n)-1} 3^a$$

Remember

- $S_n = a_0 \dfrac{1 - q^n}{1 - q}$
- $a^{\log_c(b)} = b^{\log_c(a)}$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$$

- **Base case:** $n = 4^i$
- $i = \log_4(n)$

Remember

- $S_n = a_0 \dfrac{1 - q^n}{1 - q}$
- $a^{\log_c(b)} = b^{\log_c(a)}$

$$T(n) = 3^{\log_4(n)} + \sum_{a=0}^{\log_4(n)-1} 3^a$$

$$= \sum_{a=0}^{\log_4(n)} 3^a$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$

- **Base case:** $n = 4^i$
- $i = \log_4(n)$

Remember

- $S_n = a_0 \dfrac{1 - q^n}{1 - q}$
- $a^{\log_c(b)} = b^{\log_c(a)}$

$$T(n) = 3^{\log_4(n)} + \sum_{a=0}^{\log_4(n)-1} 3^a$$

$$= \sum_{a=0}^{\log_4(n)} 3^a$$

$$= \frac{3^{\log_4(n)} - 1}{2}$$

Let's solve: $T(n) = 3T(\frac{n}{4}) + 1$

$3^i T(\frac{n}{4^i}) + 3^{i-1} + 3^{i-2} + \cdots + 1$

- **Base case:** $n = 4^i$
- $i = \log_4(n)$

Remember

- $S_n = a_0 \dfrac{1 - q^n}{1 - q}$
- $a^{\log_c(b)} = b^{\log_c(a)}$

$$
\begin{aligned}
T(n) &= 3^{\log_4(n)} + \sum_{a=0}^{\log_4(n)-1} 3^a \\
&= \sum_{a=0}^{\log_4(n)} 3^a \\
&= \frac{3^{\log_4(n)} - 1}{2} \\
&= \frac{n^{\log_4(n)}}{2} - \frac{1}{2}
\end{aligned}
$$

Hanoi Tower

- **Input:** n disks $d_1, \ldots, d_n$ with radii $r_1, \ldots, r_n$ where $r_i = r_j \Leftrightarrow i = j$, sorted on the column A.
- **Process:** Transfer all disks to column B, with help of column C.
- **Rule:** d_i can be placed onto d_j if, and only if $r_i < r_j$.

CPS 616: Algorithms

Week 3

Recurrence Relations - Part II

February 9, 2022

Onur Çağırıcı

Cost function

Algorithm: $\textsc{FindMin}(A)$

Input: An array A of n integers
Output: Minimum integer in A

$min \leftarrow A[0]$;
$i \leftarrow 0$;
while $i < n$ **do**
  if $A[i] < min$ **then**
    $min \leftarrow A[i]$;
  $i \leftarrow i + 1$;
return min

$$\begin{aligned} f(n) &= 1 + \sum_{i=0}^{n-1} 3 \\ &= 1 + 3n \end{aligned}$$

Cost function

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of all integers from 1 to n

$result \leftarrow 0;$

$i \leftarrow 0;$

for $j \leftarrow 1 \rightarrow n$ **do**

$\quad i \leftarrow i + 1;$

$\quad result \leftarrow \textsc{Add}(i, result);$

Algorithm: $\textsc{Add}(x, y)$

Input: Two integers x, y

Output: Sum of x and y

return $x + y$

$$f(n) = 2 + \sum_{i=1}^{n-1}(1 + h(n))$$

$$f(n) \in O(n)$$

Let's solve: addition of n consecutive integers

for $j \leftarrow 1 \rightarrow n$ **do**

$\quad i \leftarrow i + 1;$

$\quad result \leftarrow \textsc{Add}(i, result);$

$$\underbrace{1 + 2 + 3 + 4 + 5 + \cdots + (n-2) + (n-1)}_{\textsc{Add}(n-1)} + n$$

$$\textsc{Add}(n) \begin{cases} 1 & , n = 1 \\ \textsc{Add}(n-1) + n & , n \neq 1 \end{cases}$$

Let's solve: addition of n consecutive integers

Algorithm: $\textsc{Sum}(n)$

Input: An integer n
Output: Sum of integers from 1 to n
if $n = 1$ **then return** 1;
else return $\textsc{Sum}(n+1)+n$;

$$f(n) \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Let's solve: factorial

Algorithm: $\textsc{Fac}(n)$

Input: An integer n
Output: Product of integers from 1 to n
if $n = 1$ **then return** 1;
else return $\textsc{Fac}(n-1) \times n$;

$$f(n) \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Solving recurrence relations

- We need to have the time $f(n)$ in terms of n, NOT the function f!
- Thus, we need to solve recurrence relations.

Solving recurrence relations

- Find a recurrence relation
- Solve the recurrence relation
- Correctness proof by induction

$$\textsc{Add}(n) \begin{cases} 1 & , n = 1 \\ \textsc{Add}(n-1) + n & , n \neq 1 \end{cases}$$

$$f(n) = O(n)$$

Solving recurrence relations

- Substitution method
- Recurrence tree
- Master Theorem

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{3}) + n & , \text{otherwise} \end{cases}$$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{3}) + n & , \text{otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 2T(\frac{n}{3}) + n \\ &= 2(2T(\frac{n}{3^2}) + \frac{n}{3}) + n \\ &= 2^2T(\frac{n}{3^2}) + \frac{2}{3} + n \end{aligned}$$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{3}) + n & \text{, otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 2T(\frac{n}{3}) + n \\ &= 2(2T(\frac{n}{3^2}) + \frac{n}{3}) + n \\ &= 2^2T(\frac{n}{3^2}) + \frac{2}{3} + n \\ &= 2^2(2T(\frac{n}{3^3}) + \frac{n}{3^2}) + n \\ &= 2^3T(\frac{n}{3^3}) + \frac{2^2}{3^2}n + \frac{2}{3}n + n \\ &\vdots \end{aligned}$$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{3}) + n & \text{, otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 2T(\frac{n}{3}) + n \\ &= 2(2T(\frac{n}{3^2}) + \frac{n}{3}) + n \\ &= 2^2T(\frac{n}{3^2}) + \frac{2}{3} + n \\ &= 2^2(2T(\frac{n}{3^3}) + \frac{n}{3^2}) + n \\ &= 2^3T(\frac{n}{3^3}) + \frac{2^2}{3^2}n + \frac{2}{3}n + n \\ &\vdots \\ &= 2^iT(\frac{n}{3^i}) + n\sum_{a=0}^{i-1}\left(\frac{2}{3}\right)^a \end{aligned}$$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\frac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a}$$

- Base case:
- $i =$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\frac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a}$$

- Base case: $n = 3^i$
- $i =$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\frac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a}$$

- Base case: $n = 3^i$
- $i = \log_3(n)$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\frac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a}$$

- Base case: $n = 3^i$
- $i = \log_3(n)$

Remember

- $S_n = a_0 \dfrac{1}{1-q}$ if $q < 1$
- $a^{\log_c(b)} = b^{\log_c(a)}$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\tfrac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\tfrac{2}{3}\right)^a}$$

- Base case: $n = 3^i$
- $i = \log_3(n)$

$$= 2^{\log_3(n)} + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a$$

Remember

- $S_n = a_0 \dfrac{1}{1-q}$ if $q < 1$
- $a^{\log_c(b)} = b^{\log_c(a)}$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\tfrac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\tfrac{2}{3}\right)^a}$$

- Base case: $n = 3^i$
- $i = \log_3(n)$

$$= 2^{\log_3(n)} + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a$$

Remember

- $S_n = a_0 \dfrac{1}{1-q}$ if $q < 1$
- $a^{\log_c(b)} = b^{\log_c(a)}$

Let's solve: $T(n) = 2T(\frac{n}{3}) + n$

$$\boxed{2^i T(\tfrac{n}{3^i}) + n \sum_{a=0}^{i-1} \left(\tfrac{2}{3}\right)^a}$$

- Base case: $n = 3^i$
- $i = \log_3(n)$

Remember

- $S_n = a_0 \dfrac{1}{1-q}$ if $q < 1$
- $a^{\log_c(b)} = b^{\log_c(a)}$

$$= 2^{\log_3(n)} + n \sum_{a=0}^{i-1} \left(\frac{2}{3}\right)^a$$

$$= n^{\log_3(2)} + 3n$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n \log n$

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n \log n$

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n \log n \\
&= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n \log n \\
&= 2^2T\left(\frac{n}{2^2}\right) + n \log\frac{n}{2} + n \log n
\end{aligned}
$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n\log n \\
&= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n \\
&= 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n \\
&= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\
&= 2^3T(\frac{n}{2^3}) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\
&\vdots
\end{aligned}
$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n\log n \\
&= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n \\
&= 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n \\
&= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\
&= 2^3T(\frac{n}{2^3}) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\
&\vdots \\
&= 2^iT(\frac{n}{2^i}) + n\sum_{a=0}^{i-1}\frac{n\log n}{2^a}
\end{aligned}
$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$\boxed{2^i T(\tfrac{n}{2^i}) + n\sum_{a=0}^{i-1} \frac{n\log n}{2^a}}$$

- Base case:
- $i =$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$\boxed{2^iT(\frac{n}{2^i}) + n\sum_{a=0}^{i-1}\frac{n\log n}{2^a}}$$

- Base case: $n = 2^i$
- $i =$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$\boxed{2^i T(\frac{n}{2^i}) + n\sum_{a=0}^{i-1} \frac{n\log n}{2^a}}$$

- Base case: $n = 2^i$
- $i = \log_2(n)$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n \log n$

$$\boxed{2^i T(\frac{n}{2^i}) + n \sum_{a=0}^{i-1} \frac{n \log n}{2^a}}$$

- Base case: $n = 2^i$
- $i = \log_2(n)$

Remember

- $a^{\log_c(b)} = b^{\log_c(a)}$
- $\sum(a - b) = \sum a - \sum b$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$\boxed{2^i T(\tfrac{n}{2^i}) + n \sum_{a=0}^{i-1} \frac{n \log n}{2^a}}$$

- Base case: $n = 2^i$
- $i = \log_2(n)$

Remember

- $a^{\log_c(b)} = b^{\log_c(a)}$
- $\sum(a-b) = \sum a - \sum b$

$$= \cancelto{n}{2^{\log n} T(1)} + \sum_{a=0}^{\log n - 1} n \log \frac{n}{2^a}$$

$$= n + n^{\log_2(2)} + \dots$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n \log n$

$$\boxed{2^i T(\tfrac{n}{2^i}) + n \sum_{a=0}^{i-1} \frac{n \log n}{2^a}}$$

- Base case: $n = 2^i$
- $i = \log_2(n)$

Remember

- $a^{\log_c(b)} = b^{\log_c(a)}$
- $\sum(a-b) = \sum a - \sum b$

$$= \cancel{2^{\log n} T(1)}^{\,n} + \sum_{a=0}^{\log n - 1} n \log \frac{n}{2^a}$$

$$= n + n^{\log_2(2)} + \dots$$

$$= n + \sum_{a=0}^{\log n - 1} \log \frac{n}{2^a}$$

$$= n + n \left(\sum_{a=0}^{\log n - 1} \log n - \sum_{a=0}^{\log n - 1} \log 2^a \right)$$

$$= n + n \log n \sum_{a=0}^{\log n - 1} 1 - n \sum_{a=0}^{\log n - 1} a$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$\boxed{2^iT(\tfrac{n}{2^i}) + n\sum_{a=0}^{i-1}\frac{n\log n}{2^a}}$$

- Base case: $n = 2^i$
- $i = \log_2(n)$

Remember

- $a^{\log_c(b)} = b^{\log_c(a)}$
- $\sum(a-b) = \sum a - \sum b$

$$= \cancel{2^{\log n}T(1)} \xrightarrow{} n + \sum_{a=0}^{\log n - 1} n\log\frac{n}{2^a}$$

$$= n + n^{\log_2(2)} + \dots$$

$$= n + \sum_{a=0}^{\log n - 1}\log\frac{n}{2^a}$$

$$= n + n\left(\sum_{a=0}^{\log n - 1}\log n - \sum_{a=0}^{\log n - 1}\log 2^a\right)$$

$$= n + n\log n\sum_{a=0}^{\log n - 1}1 - n\sum_{a=0}^{\log n - 1}a$$

$$= n + \frac{n\log^2 n}{2} + \frac{n\log n}{2}$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n \log n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n \log n & , \text{otherwise} \end{cases}$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n\log n & \text{, otherwise} \end{cases}$$

From now on, $\log_2(n) = \log n$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n\log n & \text{, otherwise} \end{cases} \qquad T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

From now on, $\log_2(n) = \log n$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n\log n & \text{, otherwise} \end{cases}$$

From now on, $\log_2(n) = \log n$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n\log n \\ &= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n \\ &= 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n \end{aligned}$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n\log n & \text{, otherwise} \end{cases}$$

From now on, $\log_2(n) = \log n$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n\log n \\ &= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n \\ &= 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n \\ &= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\ &= 2^3T\left(\frac{n}{2^3}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\ &\vdots \end{aligned}$$

Let's solve: $T(n) = 2T(\frac{n}{2}) + n\log n$

$$T(n) \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n\log n & \text{, otherwise} \end{cases}$$

From now on, $\log_2(n) = \log n$

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n\log n \\
&= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n \\
&= 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n \\
&= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\
&= 2^3T\left(\frac{n}{2^3}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n \\
&\vdots \\
&= 2^iT(\frac{n}{2^i}) + n\sum_{a=0}^{i-1}\frac{n\log n}{2^a}
\end{aligned}$$

Solving recurrence relations

- Substitution method
- Recurrence tree
- Master Theorem

Solving recurrence relations – Recurrence tree

- A visual tool to unwind (expand) the recurrence relations.
- At each step, the relation is written as the sum of two terms.

Solving recurrence relations – Recurrence tree

$$\boxed{T(n) = 2T(n/2) + n\log n}$$

$$
\begin{array}{lcr}
T(n) & n\log n & = n\log n \\
T(n/2) & \frac{n}{2}\log\frac{n}{2} \quad + \quad \frac{n}{2}\log\frac{n}{2} & = n\log\frac{n}{2} \\
T(n/4) & \frac{n}{4}\log\frac{n}{4} \quad + \quad \frac{n}{4}\log\frac{n}{4} \quad + \quad \frac{n}{4}\log\frac{n}{4} \quad + \quad \frac{n}{4}\log\frac{n}{4} & = n\log\frac{n}{4} \\
 & \vdots \qquad \vdots \qquad \vdots \qquad \vdots &
\end{array}
$$

$$T(n) = \underbrace{\sum_{x=0}^{\log n} n\log\frac{n}{2^x}}_{\Theta(n\log^2 n)}$$

Solving recurrence relations – Recurrence tree

$$\begin{aligned}
\sum_{a=0}^{\log n} n \log\left(\frac{n}{2^a}\right) &= n \sum_{a=0}^{\log n} \log\left(\frac{n}{2^a}\right) = n \sum_{a=0}^{\log n} \left(\log n - \log\left(2^a\right)\right) \\
&= n \left(\sum_{a=0}^{\log n} \log n - \sum_{a=0}^{\log n} \log 2^a \right) = n \log n \sum_{a=0}^{\log n} 1 - n \sum_{a=0}^{\log n} a \\
&= n \log n (\log n + 1) - n \cdot \frac{\log n \cdot (\log n + 1)}{2} \\
&= \frac{n \log^2 n + n \log n}{2}
\end{aligned}$$

Solving recurrence relations – Recurrence tree

- If we are careful about the base cases, the recurrence tree gives the **exact** solution.
- Typically recurrence tree gives an estimate of the asymptotic notation.
- **Reminder:** after solving the recurrence relation, we must prove it using induction.

Solving recurrence relations – Recurrence tree (*)

Show that $T(n) = \Theta(n \log^2 n)$

- **First:** $T(n) = O(n \log^2 n)$
  - $M = 3$
  - $n_0 = 2$
- $T(n) \leq 3n \log^2 n$

Proof by induction

- **Base case:** $\underbrace{T(2)}_{2T\left(\frac{2}{2}\right)+2\log 2=4} \leq \underbrace{3 \times 2 \log^2 2}_{3\times2\times1=6}$
- $4 < 6$ holds.
- **Assumption:** $\forall k \in \{2, 3, \ldots, k-1\},\ T(k) \leq 3k \log^2 k$

Solving recurrence relations – Recurrence tree (*)

$$\begin{aligned}
T(k) &\leq 3k\log^2 k \\
&\leq 2 \times 3 \times \frac{k}{2}\log^2\frac{k}{2} + k\log k \\
&\leq 3k(\log k - \log 2)^2 + k\log k \\
&\leq 3k(\log k - 1)^2 + k\log k \\
&\leq 3k(\log^2 k + 1 - 2\log k) + k\log k \\
&\leq \underbrace{3k\log^2 k + 3k - 5k\log k}_{X}
\end{aligned}$$

$$\begin{aligned}
k &> 2 \\
\log k &> \log 2 \\
\log k &> 1 \\
\log k &> \frac{3}{5} \\
5k\log k &> 3k \\
0 &> 3k - 5k\log k \\
3k\log^2 k &> \underbrace{5k\log k + 3k\log^2 k}_{X}
\end{aligned}$$

Solving recurrence relations – Recurrence tree (*)

$$
\begin{aligned}
T(k) &\le 3k\log^2 k \\
&\le 2 \times 3 \times \frac{k}{2}\log^2\frac{k}{2} + k\log k \\
&\le 3k(\log k - \log 2)^2 + k\log k \\
&\le 3k(\log k - 1)^2 + k\log k \\
&\le 3k(\log^2 k + 1 - 2\log k) + k\log k \\
&\le \underbrace{3k\log^2 k + 3k - 5k\log k}_{X}
\end{aligned}
$$

$$
\begin{aligned}
k &> 2 \\
\log k &> \log 2 \\
\log k &> 1 \\
\log k &> \frac{3}{5} \\
5k\log k &> 3k \\
0 &> 3k - 5k\log k \\
3k\log^2 k &> \underbrace{5k\log k + 3k\log^2 k}_{X}
\end{aligned}
$$

$$
\boxed{T(k) \le X \le 3k\log^2 k \Rightarrow T(k) \le 3k\log^2 k}
$$

Conclusion

- Base case holds

Conclusion

- Base case holds
- Inductive hypothesis concludes the inductive step

Conclusion

- Base case holds
- Inductive hypothesis concludes the inductive step
- Therefore, $T(n) \in O(n \log^2 n)$

Solving recurrence relations – Recurrence tree (*)

Conclusion

- Base case holds
- Inductive hypothesis concludes the inductive step
- Therefore, $T(n) \in O(n \log^2 n)$
- Analogously, $T(n) \in \Omega(n \log^2 n)$

Conclusion

- Base case holds
- Inductive hypothesis concludes the inductive step
- Therefore, $T(n) \in O(n \log^2 n)$
- Analogously, $T(n) \in \Omega(n \log^2 n)$

Hence, $T(n) \in \Theta(n \log^2 n) \quad \square$

Recurrence tree: Example 1

$$\boxed{T(n) = 2T(n/2) + n^2}$$

Recurrence tree: Example 1

$$\boxed{T(n) = 2T(n/2) + n^2}$$

$$\begin{array}{lccccccccl}
T(n) & & & & n \log n & & & & & = n^2 \\
& & \swarrow & & & & \searrow & & & \\
T(n/2) & & \left(\frac{n}{2}\right)^2 & & + & & \left(\frac{n}{2}\right)^2 & & & = \frac{n^2}{2} \\
& \swarrow & & \searrow & & \swarrow & & \searrow & & \\
T(n/4) & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & & = \frac{n^2}{4} \\
& \vdots & & \vdots & & \vdots & & \vdots & &
\end{array}$$

Recurrence tree: Example 1

$$\boxed{T(n) = 2T(n/2) + n^2}$$

$$
\begin{array}{ccccccccc}
T(n) & & & & n\log n & & & & = n^2 \\
 & & \swarrow & & & & \searrow & & \\
T(n/2) & & \left(\frac{n}{2}\right)^2 & & + & & \left(\frac{n}{2}\right)^2 & & = \frac{n^2}{2} \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
T(n/4) & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & = \frac{n^2}{4} \\
 & \vdots & & \vdots & & \vdots & & \vdots &
\end{array}
$$

$$T(n) = n^2 \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log n}}\right)$$

Recurrence tree: Example 1

$$\boxed{T(n) = 2T(n/2) + n^2}$$

$$\begin{array}{cccccccc}
T(n) & & & n\log n & & & & = n^2 \\
T(n/2) & \left(\frac{n}{2}\right)^2 & & + & & \left(\frac{n}{2}\right)^2 & & = \frac{n^2}{2} \\
T(n/4) & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & = \frac{n^2}{4} \\
 & \vdots & & \vdots & \vdots & & \vdots &
\end{array}$$

$$T(n) = n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log n}}\right)$$

$$q < 1 \Rightarrow S_n = a_0\frac{1}{1-q} \Rightarrow 1\frac{1}{1-\frac{1}{2}} = 2$$

Recurrence tree: Example 1

$$\boxed{T(n) = 2T(n/2) + n^2}$$

$$
\begin{array}{lccccccccr}
T(n) & & & & n \log n & & & & & = n^2 \\
T(n/2) & & \left(\frac{n}{2}\right)^2 & & + & & \left(\frac{n}{2}\right)^2 & & & = \frac{n^2}{2} \\
T(n/4) & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & + & \left(\frac{n}{4}\right)^2 & & = \frac{n^2}{4} \\
 & \vdots & & \vdots & & \vdots & & \vdots & &
\end{array}
$$

$$T(n) = n^2 \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log n}}\right)$$

$$q < 1 \Rightarrow S_n = a_0 \frac{1}{1-q} \Rightarrow 1 \frac{1}{1-\frac{1}{2}} = 2$$

$$\therefore T(n) = 2n^2$$

Recurrence tree: Example 2

$$\boxed{T(n) = T(n/3) + T(2n/3) + n}$$

Recurrence tree: Example 2

$$\boxed{T(n) = T(n/3) + T(2n/3) + n}$$

$$T(n) \qquad n \qquad = n$$

$$T(n/3), t(2n/3) \qquad \frac{n}{3} \quad + \quad \frac{2n}{3} \qquad = n$$

$$T(\tfrac{n}{9}), T(\tfrac{2n}{9}), T(\tfrac{2n}{9}), T(\tfrac{4n}{9}) \qquad \frac{n}{9} \quad + \quad \frac{2n}{9} \quad + \quad \frac{2n}{9} \quad + \quad \frac{4n}{9} \qquad = n$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

Recurrence tree: Example 2

$$\boxed{T(n) = T(n/3) + T(2n/3) + n}$$

$$
\begin{array}{lcccccccl}
T(n) & & & & n & & & & = n \\
T(n/3), t(2n/3) & & \frac{n}{3} & & + & & \frac{2n}{3} & & = n \\
T(\frac{n}{9}), T(\frac{2n}{9}), T(\frac{2n}{9}), T(\frac{4n}{9}) & \frac{n}{9} & + & \frac{2n}{9} & + & \frac{2n}{9} & + & \frac{4n}{9} & = n \\
& \vdots & & \vdots & & \vdots & & \vdots &
\end{array}
$$

$$\frac{n}{3^i} = 1 \Rightarrow i = \log_3 n$$

Recurrence tree: Example 2

$$\boxed{T(n) = T(n/3) + T(2n/3) + n}$$

$$
\begin{array}{rccccccccc}
T(n) & & & & n & & & & & = n \\
T(n/3), t(2n/3) & & \frac{n}{3} & & + & & \frac{2n}{3} & & & = n \\
T(\frac{n}{9}), T(\frac{2n}{9}), T(\frac{2n}{9}), T(\frac{4n}{9}) & \frac{n}{9} & + & \frac{2n}{9} & + & \frac{2n}{9} & + & \frac{4n}{9} & & = n \\
 & \vdots & & \vdots & & \vdots & & \vdots & &
\end{array}
$$

$$\frac{n}{3^i} = 1 \Rightarrow i = \log_3 n$$

$$T(n) = \underbrace{n + n + \cdots + n}_{i = \log_3 n}$$

Up to now

- Analysis of an algorithm
  - Cost function
  - Asymptotic notation
  - Finding the cost of recursive algorithms
- These notions let us compare different algorithms by measuring the time complexity of each one

Algorithmic techniques

- Greedy algorithms
- Divide and conquer
- Dynamic programming
- Randomized algorithms

Reference

- 16.1, 16.2, 16.3
- 23.1, 23.2

Optimization problem

- Finding the **best** solution for a given problem, in terms of **cost or benefit**, while there exist several feasible solutions.

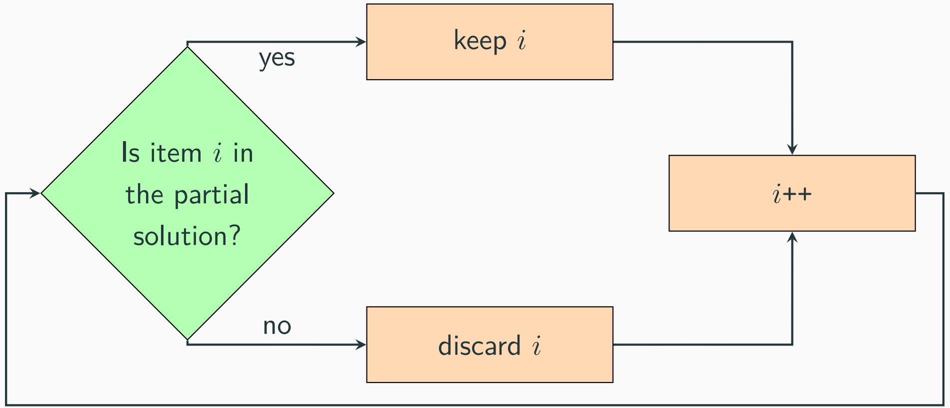
Optimization problem

- Finding the **best** solution for a given problem, in terms of **cost or benefit**, while there exist several feasible solutions.
- Greedy algorithms

Optimization problem

- Finding the **best** solution for a given problem, in terms of **cost or benefit**, while there exist several feasible solutions.
- Greedy algorithms
  - Principle is local optimization
  - At each step, select a part of the solution based on selection function
  - There is **no backtracking** in greedy algorithms

How do greedy algorithms work?



0-1 Knapsack problem

- Given n items;
- Each item i with value c_i and weight w_i.
- Goal is to fill a backpack with capacity m such that the bag contains maximum possible value

The items cannot be carried partially

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

- Examine all **feasible** subsets

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

- Examine all **feasible** subsets
- Take the **optimal** solution

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

- Examine all **feasible** subsets
- Take the **optimal** solution
- Time complexity: $O(2^n)$

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

- Examine all **feasible** subsets
- Take the **optimal** solution
- Time complexity: $O(2^n)$

Instead of using brute-force, we make a

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy

1. add largest remaining item

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy

1. add largest remaining item
2. add most valuable remaining item

0-1 Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy

1. add largest remaining item
2. add most valuable remaining item
3. add densest remaining item

Properties of a greedy algorithm

- **Optimal substructure:** an optimal solution includes optimal sub-solutions
- **Greedy choice properties:** choosing a locally optimal choice leads to a globally optimal solution

Fractional Knapsack problem

- Given n items;
- Each item i with value c_i and weight w_i.
- Goal is to fill a backpack with capacity m such that the bag contains maximum possible value

This time, the items can be broken apart and carried partially

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #1

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #1

1. Sort the items with respect to their weights (w_i)

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #1

1. Sort the items with respect to their weights (w_i)
2. Place the item i with the highest w_i in the bag **if the bag has space**

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #1

1. Sort the items with respect to their weights (w_i)
2. Place the item i with the highest w_i in the bag **if the bag has space**
3. If the bag does not have sufficient space, then place a **fraction** of the item i

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #1

1. Sort the items with respect to their weights (w_i)
2. Place the item i with the highest w_i in the bag **if the bag has space**
3. If the bag does not have sufficient space, then place a **fraction** of the item i
4. Repeat until the bag is full

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

$$2 + 2\left(\frac{2}{5}\right) = 2.8$$

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy #2

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #2

1. Sort the items with respect to their values (c_i)

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy #2

1. Sort the items with respect to their values (c_i)
2. Place the item i with the highest c_i in the bag **if the bag has space**

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy #2

1. Sort the items with respect to their values (c_i)
2. Place the item i with the highest c_i in the bag **if the bag has space**
3. If the bag does not have sufficient space, then place a **fraction** of the item i

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #2

1. Sort the items with respect to their values (c_i)
2. Place the item i with the highest c_i in the bag **if the bag has space**
3. If the bag does not have sufficient space, then place a **fraction** of the item i
4. Repeat until the bag is full

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

$$2 + 2 + 3\left(\frac{2}{8}\right) = 4.75$$

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy #3

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #3

1. Sort the items with respect to their ratios of weight-to-value (c_i/w_i)

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	\$1
2	8	\$2
3	4	\$1
4	2	\$2
5	5	\$2

Possible greedy strategy #3

1. Sort the items with respect to their ratios of weight-to-value (c_i/w_i)
2. Place the item i with the highest c_i/w_i in the bag **if the bag has space**

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #3

1. Sort the items with respect to their ratios of weight-to-value (c_i/w_i)
2. Place the item i with the highest c_i/w_i in the bag **if the bag has space**
3. If the bag does not have sufficient space, then place a **fraction** of the item i

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value
1	3	$1
2	8	$2
3	4	$1
4	2	$2
5	5	$2

Possible greedy strategy #3

1. Sort the items with respect to their ratios of weight-to-value (c_i/w_i)
2. Place the item i with the highest c_i/w_i in the bag **if the bag has space**
3. If the bag does not have sufficient space, then place a **fraction** of the item i
4. Repeat until the bag is full

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value	V/W
1	3	\$1	1/3
2	8	\$2	1/4
3	4	\$1	1/4
4	2	\$2	1
5	5	\$2	2/5

Fractional Knapsack problem – $m = 10$

Item #	Weight	Value	V/W
1	3	\$1	1/3
2	8	\$2	1/4
3	4	\$1	1/4
4	2	\$2	1
5	5	\$2	2/5

$$2(1) + 5\left(\frac{2}{5}\right) + 3\left(\frac{1}{3}\right) = 5$$

Why use greedy algorithms?

Question: What would be the reasons to implement a greedy algorithm?

Why use greedy algorithms?

Question: What would be the reasons to implement a greedy algorithm?

- Fast

Why use greedy algorithms?

Question: What would be the reasons to implement a greedy algorithm?

- Fast
- Easy to implement

Why use greedy algorithms?

Question: What would be the reasons to implement a greedy algorithm?

- Fast
- Easy to implement
- Easy to understand

Why use greedy algorithms?

Question: What would be the reasons to implement a greedy algorithm?

- Fast
- Easy to implement
- Easy to understand

However,

Why use greedy algorithms?

Question: What would be the reasons to implement a greedy algorithm?

- Fast
- Easy to implement
- Easy to understand

However,

greedy algorithms do not always yield the optimum solution.

Why use greedy algorithms?

Question: What if a greedy algorithm does not give the optimum solution?

Why use greedy algorithms?

Question: What if a greedy algorithm does not give the optimum solution?

- If we can prove that the greedy algorithm can somewhat find a "close" solution to the optimum in the worst case,

Why use greedy algorithms?

Question: What if a greedy algorithm does not give the optimum solution?

- If we can prove that the greedy algorithm can somewhat find a "close" solution to the optimum in the worst case,
  - then we have a **approximation algorithm**

Why use greedy algorithms?

Question: What if a greedy algorithm does not give the optimum solution?

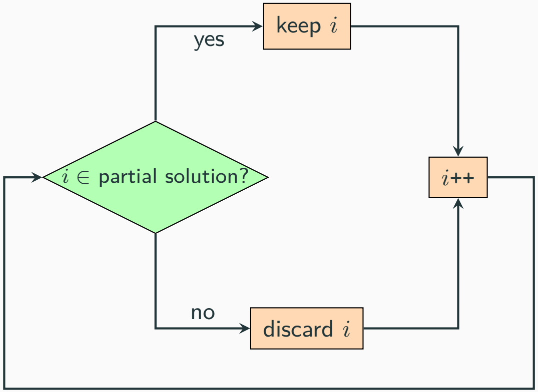
- If we can prove that the greedy algorithm can somewhat find a "close" solution to the optimum in the worst case,
  - then we have a **approximation algorithm**
- If there is no such a proof,
  - then we have a **heuristic approach**

Activity selection problem

- Given n tasks $\{t_1, \ldots, t_n\}$;
  - no t_i and t_j can share the resources
  - each task has a starting time s_i
  - each task has a finishing time f_i
- Goal is to execute the maximum number of activities

Activity selection problem

Tasks	Start	Finish
1	2	13
2	6	10
3	5	7
4	0	6
5	8	11
6	3	5
7	1	4
8	8	12
9	12	14
10	5	9



Activity selection problem – Greedy solution

We want to execute the maximum number of possible tasks among the tasks $t_1, t_2, \ldots, t_n$ that are compatible (non-overlapping).

1. Sort the set of tasks each task based on their finishing time.
   - Let t_{min} be a task with the minimum finishing time.
   - Let s_{min} and f_{min} be the starting time, and the finishing time of t_{min}, respectively.
2. Remove all t_i such that $s_i > s_{min}$ and $f_i < f_{min}$ (i.e., tasks that overlap with t_{min}.
3. Include the task t_{min} in the solution.
4. Remove t_{min}.
5. If all tasks are removed, then exit.
6. If there exist unprocessed tasks, then go to Step 1.

Activity selection problem – Greedy solution

Tasks	Start	Finish
1	2	13
2	6	10
3	5	7
4	0	6
5	8	11
6	3	5
7	1	4
8	8	12
9	12	14
10	5	9

One processor, n tasks

- We have one processor, and n tasks $t_1, \ldots, t_n$ to execute.

One processor, n tasks

- We have one processor, and n tasks $t_1, \ldots, t_n$ to execute.
- There is a deadline d_i for each task t_i where $1 \leq i \leq n$.

One processor, n tasks

- We have one processor, and n tasks $t_1, \ldots, t_n$ to execute.
- There is a deadline d_i for each task t_i where $1 \leq i \leq n$.
- If a task is completed after its deadline, i.e., $f_i > d_i$, then a penalty p_i is applied.

One processor, n tasks

- We have one processor, and n tasks $t_1, \ldots, t_n$ to execute.
- There is a deadline d_i for each task t_i where $1 \leq i \leq n$.
- If a task is completed after its deadline, i.e., $f_i > d_i$, then a penalty p_i is applied.
- Goal is to complete a schedule with minimum penalty.

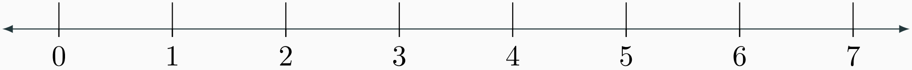
One processor, n tasks

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50

- Sort the tasks ascendingly with respect to their penalties.
- Place each task i before its deadline d_i if there is available slot.
  - Pick the rightmost task if there are multiple tasks.
- Otherwise, the task i needs to be scheduled with a delay.
- Thus, we schedule it after we process the rest of the tasks.

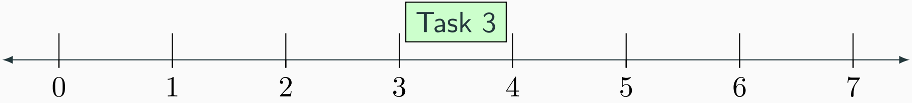
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



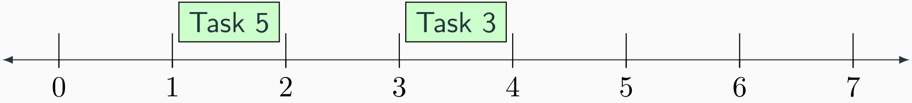
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



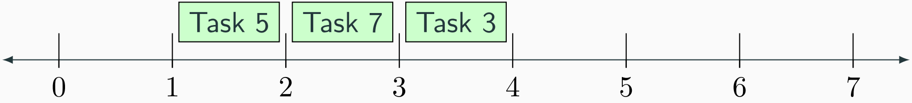
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



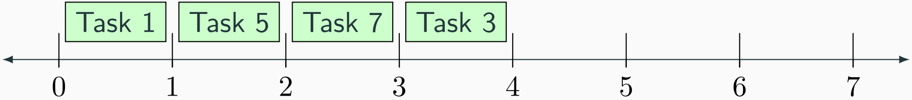
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



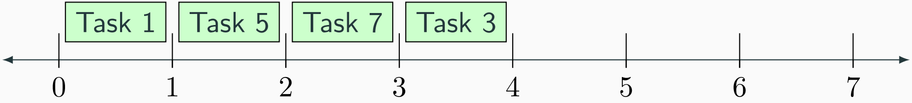
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



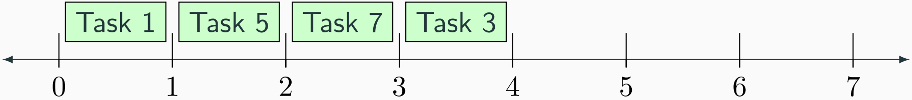
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



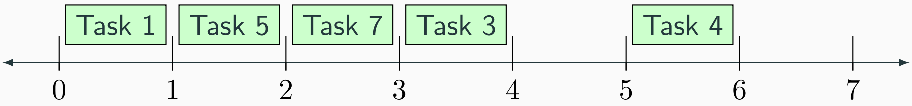
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



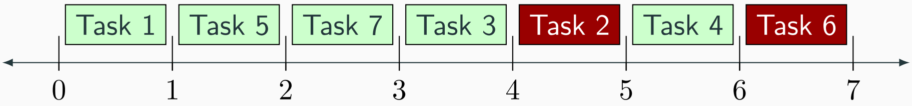
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



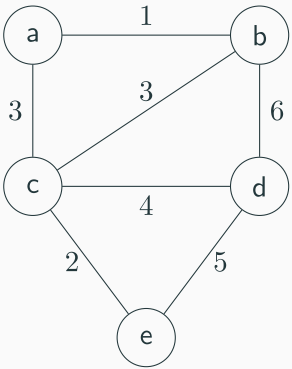
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Start	Finish
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50

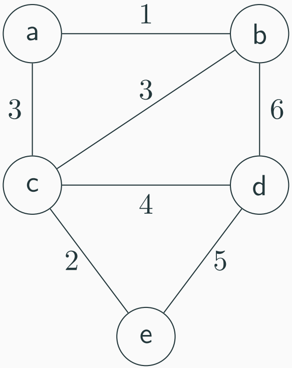


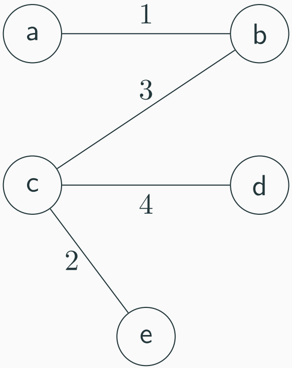
Minimum spanning tree

Minimum spanning tree



Minimum spanning tree





Prim's Algorithm

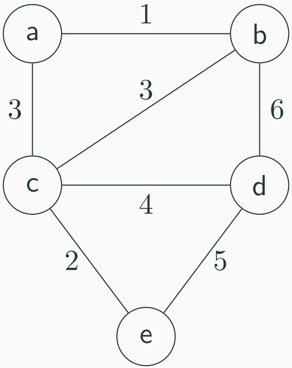
- Start with an empty set F of edges.
- Start with an empty set Y of vertices.
- At each step, greedily add the vertices to Y along with the corresponding edges.

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G

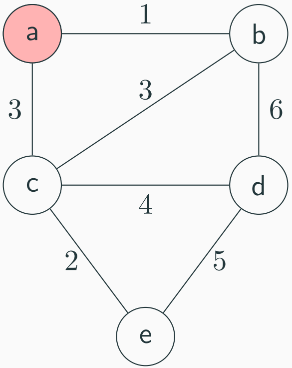
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
    Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
    $F \leftarrow F \cup \{uv\}$;
    $Y \leftarrow Y \cup \{v\}$;
return F;



Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
    Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
    $F \leftarrow F \cup \{uv\}$;
    $Y \leftarrow Y \cup \{v\}$;
return F;



$Y = \{a\}$

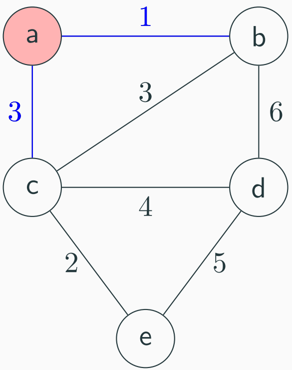
$F = \{\}$

$V \setminus Y = \{\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
  Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
  $F \leftarrow F \cup \{uv\}$;
  $Y \leftarrow Y \cup \{v\}$;
return F;



$Y = \{a\}$

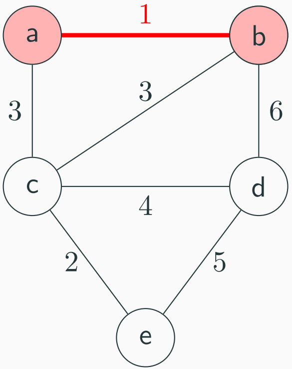
$F = \{\}$

$V \setminus Y = \{b, c, d, e\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
  Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
  $F \leftarrow F \cup \{uv\}$;
  $Y \leftarrow Y \cup \{v\}$;
return F;



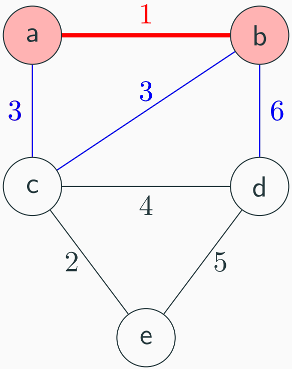
$$Y = \{a, b\} \qquad F = \{ab\}$$

$$V \setminus Y = \{c, d, e\}$$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
    Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
    $F \leftarrow F \cup \{uv\}$;
    $Y \leftarrow Y \cup \{v\}$;
return F;



$Y = \{a, b\}$ $F = \{ab\}$

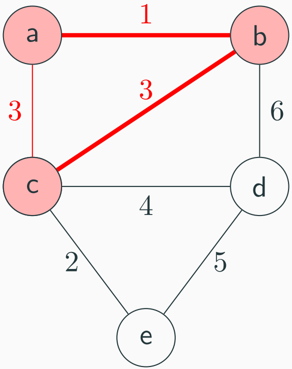
$V \setminus Y = \{c, d, e\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G

$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
  Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
  $F \leftarrow F \cup \{uv\}$;
  $Y \leftarrow Y \cup \{v\}$;
return F;



$Y = \{a, b, c\}$

$F = \{ab, bc\}$

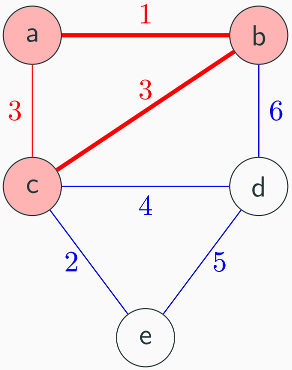
$V \setminus Y = \{d, e\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G

$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
  Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
  $F \leftarrow F \cup \{uv\}$;
  $Y \leftarrow Y \cup \{v\}$;
return F;



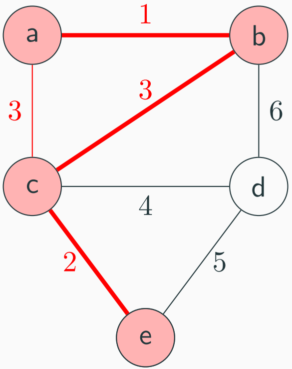
$Y = \{a, b, c\}$ $F = \{ab, bc\}$

$V \setminus Y = \{d, e\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
    Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
    $F \leftarrow F \cup \{uv\}$;
    $Y \leftarrow Y \cup \{v\}$;
return F;



$Y = \{a, b, c, e\}$ $\qquad$ $F = \{ab, bc, ce\}$

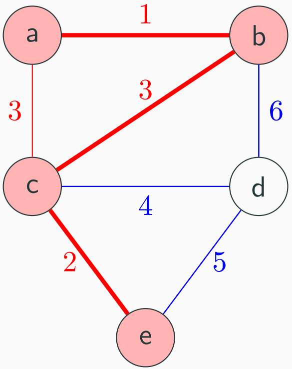
$V \setminus Y = \{d\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
  Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
  $F \leftarrow F \cup \{uv\}$;
  $Y \leftarrow Y \cup \{v\}$;

return F;



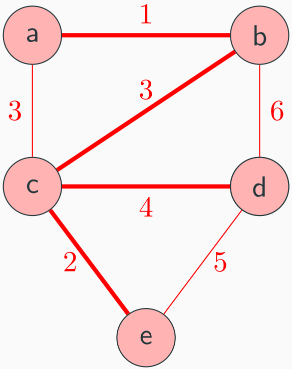
$Y = \{a, b, c, e\}$ $F = \{ab, bc, ce\}$

$V \setminus Y = \{d\}$

Prim's Algorithm

Algorithm: $\textsc{Prim}(G)$

Input: A simple, undirected graph G
Output: A minimum spanning tree of G
$F \leftarrow \emptyset$;
$Y \leftarrow \{a\}$;
while $Y \neq F$ **do**
  Let uv be the lowest cost edge such that $u \in Y$ and $v \in V \setminus Y$;
  $F \leftarrow F \cup \{uv\}$;
  $Y \leftarrow Y \cup \{v\}$;
return F;



$$Y = \{a, b, c, e, d\} \quad F = \{ab, bc, ce, cd\}$$

$$V \setminus Y = \{\}$$

CPS 616: Algorithms

Week 3

Recurrence Relations - Part III

February 15, 2022

Onur Çağırıcı

Addition of n consecutive integers

Algorithm: $\textsc{Sum}(n)$

Input: An integer n

Output: Sum of integers from 1 to n

if $n = 1$ **then return** 1;

else return $\textsc{Sum}(n+1) + n$;

$$f(n) = \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n \neq 1 \end{cases}$$

Solving recurrence relations

- We need to have the time $f(n)$ in terms of n, NOT the function f!
- Thus, we need to solve recurrence relations.

Solving recurrence relations

- Substitution method
- Recurrence tree
- Master Theorem

Solving recurrence relations – Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

Solving recurrence relations – Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & ,n > c \\ d & ,n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

Example: $T(n) = T\left(\frac{n}{2}\right) + 1$ or $T(n) = 2T\left(\frac{n}{2}\right) + n$

Solving recurrence relations – Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

Example: $T(n) = T\left(\frac{n}{2}\right) + 1$ or $T(n) = 2T\left(\frac{n}{2}\right) + n$

1. $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a}\right)$

Solving recurrence relations – Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

Example: $T(n) = T\left(\frac{n}{2}\right) + 1$ or $T(n) = 2T\left(\frac{n}{2}\right) + n$

1. $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a}\right)$
2. $f(n) \in O\left(n^{\log_b a}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$

Solving recurrence relations – Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

Example: $T(n) = T\left(\frac{n}{2}\right) + 1$ or $T(n) = 2T\left(\frac{n}{2}\right) + n$

1. $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a}\right)$
2. $f(n) \in O\left(n^{\log_b a}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$
3. $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $\exists \kappa > 0,\ n_0 > 0$ s.t. $\forall n > n_0,\ af(n/b) \leq \kappa f(n) \quad \Rightarrow \quad T(n) \in \Theta(f(n))$

Solving recurrence relations – Master theorem

$$aT\overbrace{\left(\frac{n}{b}\right)}^{A} + \overbrace{f(n)}^{B}$$

1. The overall cost is dominated by the recursive part: $A > B$
2. The cost of the recursive part and the local part are asymptotically equivalent: $A = B$
3. The overall cost is dominated by the local part: $A < B$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

Let's solve: $f(n) \in O\left(n^{\log_b a-\epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a =$

Let's solve: $f(n) \in O\left(n^{\log_b a-\epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b =$

Let's solve: $f(n) \in O\left(n^{\log_b a-\epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b = 2$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b = 2$
  - $f(n) =$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b = 2$
  - $f(n) = n^2$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a =$

Let's solve: $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a = \log_2 7 = 2.807$

Let's solve: $f(n) \in O\left(n^{\log_b a-\epsilon}\right) \implies T(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 7$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a = \log_2 7 = 2.807$

$$\begin{aligned} f(n) &= n^2 \\ &\in O(n^{2.807-\epsilon}) \\ &\in \Theta(n^{2.807}) \end{aligned}$$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a =$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b =$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b = 2$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b = 2$
  - $f(n) =$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b = 2$
  - $f(n) = 1$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b = 2$
  - $f(n) = 1$
  - $\log_b a =$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b = 2$
  - $f(n) = 1$
  - $\log_b a = \log_2 1$

Let's solve: $f(n) \in \Theta\left(n^{\log_b a}\right) \implies T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 1$
  - $b = 2$
  - $f(n) = 1$
  - $\log_b a = \log_2 1$

$$\begin{aligned} f(n) &= 1 \\ &\in \Theta\left(n^{\log_b a}\right) \\ &\in \Theta(n^0) \\ &\in \Theta(1) \\ &\in \Theta\left(n^{\log_b a} \log n\right) \\ &\in \Theta(\log n) \end{aligned}$$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right);\ \exists\kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right);\ \exists \kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a =$
  - $b =$
  - $f(n) =$
  - $\log_b a =$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right)$; $\exists \kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 2$
  - $b =$
  - $f(n) =$
  - $\log_b a =$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$; $\exists \kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 2$
  - $b = 2$
  - $f(n) =$
  - $\log_b a =$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right);\ \exists\kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 2$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a =$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right);\ \exists\kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 2$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a = \log_2 2$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right);\ \exists \kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 2$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a = \log_2 2$

$$\begin{aligned} f(n) &= n^2 \\ &\in \Omega\left(n^{\cancelto{1}{\log_b a}+\epsilon}\right) \end{aligned}$$

Let's solve: $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right);\ \exists \kappa > 0,\ n_0 > 0 \mid n > n_0,\ af(n/b) \leq \kappa f(n)$

$$T(n) = 2T\left(\tfrac{n}{2}\right) + n^2$$

- $T(n) = aT\left(\frac{n}{2}\right) + f(n)$
  - $a = 2$
  - $b = 2$
  - $f(n) = n^2$
  - $\log_b a = \log_2 2$

$$\begin{aligned} f(n) &= n^2 \\ &\in \Omega\left(n^{\cancelto{1}{\log_b a}+\epsilon}\right) \end{aligned}$$

$$\begin{aligned} a \cdot f\left(\frac{n}{b}\right) &= 2 \cdot f\left(\frac{n}{2}\right) \\ &= 2 \cdot \left(\frac{n}{2}\right)^2 \\ &= \frac{n^2}{2} \\ &\leq \frac{1}{2} \cdot f(n) \end{aligned}$$

therefore

$$\begin{aligned} T(n) &\in \Theta\left(f(n)\right) \\ &= \Theta(n^2) \end{aligned}$$

Master theorem – Not always applicable

$$T(n) = 2T(n/2) + n \log n$$

- $a =$
- $b =$
- $f(n) =$
- $\log_b a =$

Master theorem – Not always applicable

$$T(n) = 2T(n/2) + n \log n$$

$$T(n) = aT\left(\frac{n}{2}\right) + f(n)$$

- $a =$
- $b =$
- $f(n) =$
- $\log_b a =$

Master theorem – Not always applicable

$$T(n) = 2T(n/2) + n \log n$$

$$T(n) = aT\left(\frac{n}{2}\right) + f(n)$$

- $a = 2$
- $b =$
- $f(n) =$
- $\log_b a =$

Master theorem – Not always applicable

$T(n) = 2T(n/2) + n \log n$

$T(n) = aT\left(\frac{n}{2}\right) + f(n)$

- $a = 2$
- $b = 2$
- $f(n) =$
- $\log_b a =$

Master theorem – Not always applicable

$T(n) = 2T(n/2) + n \log n$

$T(n) = aT\left(\frac{n}{2}\right) + f(n)$

- $a = 2$
- $b = 2$
- $f(n) = n^2$
- $\log_b a =$

Master theorem – Not always applicable

$T(n) = 2T(n/2) + n \log n$

$T(n) = aT\left(\frac{n}{2}\right) + f(n)$

- $a = 2$
- $b = 2$
- $f(n) = n^2$
- $\log_b a = \log_2 2$

Master theorem – Not always applicable

$T(n) = 2T(n/2) + n\log n$

$T(n) = aT\left(\frac{n}{2}\right) + f(n)$

- $a = 2$
- $b = 2$
- $f(n) = n^2$
- $\log_b a = \log_2 2$

$$\begin{aligned} f(n) &= n\log n \\ &\notin O\left(n^{1+\epsilon}\right) \\ &\notin \Theta\left(n^{1}\right) \\ &\notin \Omega\left(n^{1+\epsilon}\right) \end{aligned}$$

Master theorem – Not always applicable

Recurrence relations are not always in the form which allows Master theorem to be used

$$\begin{aligned} T(n) &= aT\left(\frac{n}{b}\right) + f(n) \\ &= \left(\frac{n}{5}\right) + 7T\left(\frac{n}{10}\right) + n \end{aligned}$$

Master theorem – Not always applicable

Recurrence relations are not always in the form which allows Master theorem to be used

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{b}\right) + f(n) \\
&= \left(\frac{n}{5}\right) + 7T\left(\frac{n}{10}\right) + n
\end{aligned}
$$

The above recurrence can be solved exactly with other approaches

Master theorem – Simpler version

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $a \geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

$$\begin{aligned}
a \cdot f\left(\tfrac{n}{b}\right) &= \kappa \cdot f(n) \text{ for some } \kappa > 1 &&\Rightarrow\quad T(n) \in \Theta\left(n^{\log_b a}\right) \\
a \cdot f\left(\tfrac{n}{b}\right) &= f(n) &&\Rightarrow\quad T(n) \in \Theta\left(f(n) \log n\right) \\
a \cdot f\left(\tfrac{n}{b}\right) &= \kappa \cdot f(n) \text{ for some } \kappa < 1 &&\Rightarrow\quad T(n) \in \Theta\left(f(n)\right)
\end{aligned}$$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a =$
- $b =$
- $f(n) =$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b =$
- $f(n) =$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $f(n) =$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $f(n) = n^2$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $f(n) = n^2$

$$a \cdot f\left(\frac{n}{b}\right) = 7f\left(\frac{n}{2}\right) \qquad = 7 \cdot (n/2)^2 = \frac{7n^2}{4} \qquad = \frac{7}{4} \cdot f(n)$$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $f(n) = n^2$

$$a \cdot f\left(\frac{n}{b}\right) = 7f\left(\frac{n}{2}\right) \qquad = 7 \cdot (n/2)^2 = \frac{7n^2}{4} \qquad = \frac{7}{4} \cdot f(n)$$

Remember: $a \cdot d\left(\frac{n}{b}\right) = \kappa f(n)$ for some $\kappa > 1 \Rightarrow t(n) \in \Theta\left(n^{\log_b a}\right)$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $f(n) = n^2$

$$a \cdot f\left(\frac{n}{b}\right) = 7f\left(\frac{n}{2}\right) \qquad = 7 \cdot (n/2)^2 = \frac{7n^2}{4} \qquad = \frac{7}{4} \cdot f(n)$$

Remember: $a \cdot d\left(\frac{n}{b}\right) = \kappa f(n)$ for some $\kappa > 1 \Rightarrow t(n) \in \Theta\left(n^{\log_b a}\right)$

$\frac{7}{4} > T(n) \in \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_2 7}\right)$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a =$
- $b =$
- $\log_b a =$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b =$
- $\log_b a =$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $\log_b a =$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $\log_b a = 2.807$

Master theorem – Simpler version

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

- $a = 7$
- $b = 2$
- $\log_b a = 2.807$

$$\begin{aligned} f(n) &= n^2 \\ f(n) &\in O(n^{2.807-\epsilon}) \\ &\in \Theta(n^{\log_b a}) \\ &\in \Theta(n^{2.807}) \end{aligned}$$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a =$
- $b =$
- $f(n) =$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b =$
- $f(n) =$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b = 2$
- $f(n) =$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b = 2$
- $f(n) = \log n$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b = 2$
- $f(n) = \log n$

$$a \cdot f\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) \qquad = 4\log n - 4\log_2 2 \neq \kappa \log n \text{ for any } \kappa$$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b = 2$
- $f(n) = \log n$

$$a \cdot f\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) \qquad = 4\log n - 4\log_2 2 \neq \kappa \log n \text{ for any } \kappa$$

Remember: $\log \frac{a}{b} = \log a - \log b$

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b = 2$
- $f(n) = \log n$

$$a \cdot f\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) \qquad = 4\log n - 4\log_2 2 \neq \kappa \log n \text{ for any } \kappa$$

Remember: $\log \frac{a}{b} = \log a - \log b$

Simple version does not apply

Master theorem – Simpler version

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

- $a = 4$
- $b = 2$
- $f(n) = \log n$

$$a \cdot f\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) \qquad = 4\log n - 4\log_2 2 \neq \kappa \log n \text{ for any } \kappa$$

Remember: $\log \frac{a}{b} = \log a - \log b$

Simple version does not apply

What about the raw Master Theorem?

Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$a = 4,\ b = 2,\ f(n) = \log n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = \log n \in ?$$

Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$a = 4,\ b = 2,\ f(n) = \log n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = \log n \in ?$$

Example: $T(n) = T\left(\frac{n}{2}\right) + 1$ or $T(n) = 2T\left(\frac{n}{2}\right) + n$

Master theorem

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & , n > c \\ d & , n \leq c \end{cases}$$

where $\geq 1,\ b > 1,\ c \geq 1,\ d \geq 0$

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$a = 4,\ b = 2,\ f(n) = \log n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = \log n \in ?$$

Example: $T(n) = T\left(\frac{n}{2}\right) + 1$ or $T(n) = 2T\left(\frac{n}{2}\right) + n$

1. $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a}\right)$ ✓
2. ~~$f(n) \in O\left(n^{\log_b a}\right) \quad \Rightarrow \quad T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$~~
3. ~~$f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $\exists \kappa > 0,\ n_0 > 0$ s.t. $\forall n > n_0,\ af(n/b) \leq \kappa f(n)$ $\quad \Rightarrow \quad T(n) \in \Theta(f(n))$~~