Asymptotic Analysis of Algorithms

Analysing Algorithm Efficiency

Things we might want to know:

- How efficient is a given algorithm?
- What sized problems can be solved within a reasonable time?
- Is some new algorithm really more efficient that the existing one?
- Which of two possible algorithms is best for my particular use?

We'll look at ways to answer questions like these

Experimental Study

Measuring running time of an algorithm through experimentation

- Implement algorithm
- Choose appropriate inputs
- Run program on all inputs
- Plot results
- Determine rate at which time increases as input grows

Experimental Study

Can have signicant advantages:

- Can give very realistic impression of actual time
- Can be useful when comparing two candidate algorithms
- Subtle differences in running time may emerge
- Useful when software, hardware and possible inputs can be established

Experimental Study (cont.)

Variety of potential problems:

- Can be time-consuming to implement algorithms effectively
- Can be time-consuming to run (slow) algorithm on (large) inputs
- May not have good understanding of the context in which algorithm will be deployed
  - software used for implementation
  - hardware on which it will run
  - range of inputs on which it will run
- Comparing algorithms requires comparable implementations

Asymptotic Analysis

Look at how running time increases as input size grows

- A rough classification of the rate of growth
- Concerned with the (very) long term growth rate

Running time as a function

- Problem size is n
- Running time of algorithm referred to as $t(n)$
- $t(n)$ is the number of steps that an input of size n takes

Asymptotic Analysis (cont.)

Measure running time in terms of number of “steps”:

- lines of pseudocode
- lines of Java
- lines of assembly code

Size of step doesn’t matter when performing asymptotic analysis

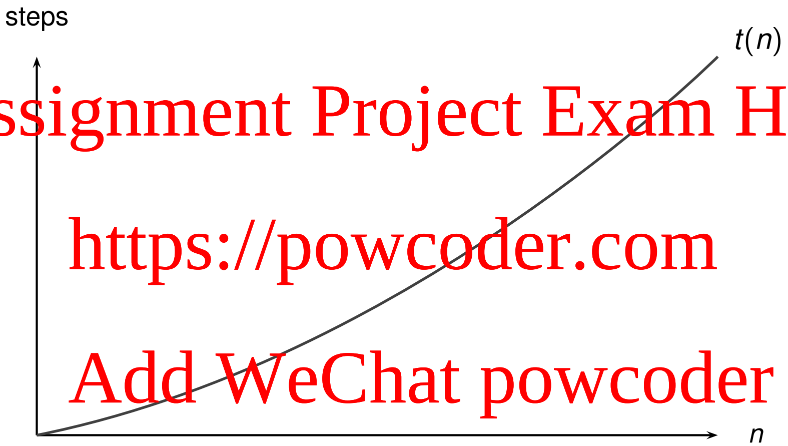
Asymptotic Bounds

Asymptotic analysis of efficiency involves determining **bounds**

- Upper bound $O(.)$
- Lower bound $\Omega(.)$
- Tight bound $\Theta(.)$

We'll begin by considering each bound graphically

Asymptotic Upper Bound: Example



Suppose this is a plot of the running time $t(n)$ of our algorithm

Questions for you

What is n a measure of for the following problems?

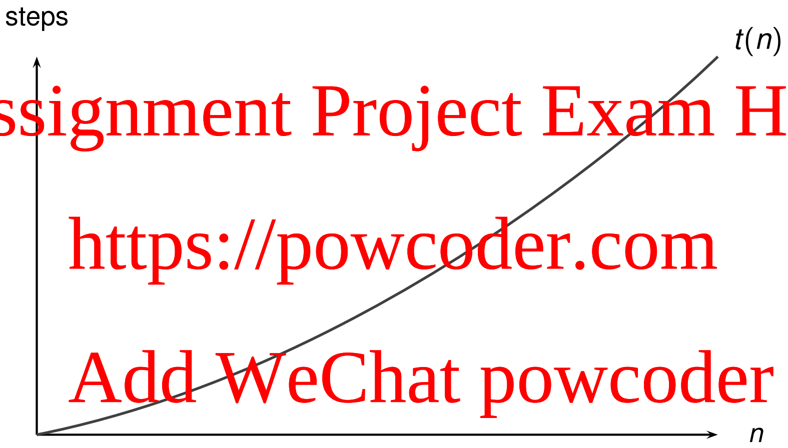
- Finding a stable match
- Sorting a sequence of numbers
- Searching a list for an given value
- Performing depth-first search of a graph

Questions for you

What is n a measure of for the following problems?

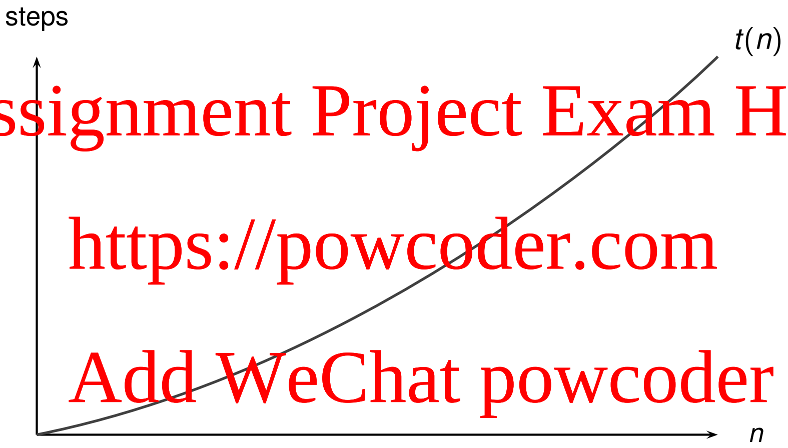
- Finding a stable match
  The number of men
- Sorting a sequence of numbers
  The number of numbers
- Searching a list for an given value
  The length of the list
- Performing depth-first search of a graph
  The number of vertices and the number of edges

Asymptotic Upper Bound: Example



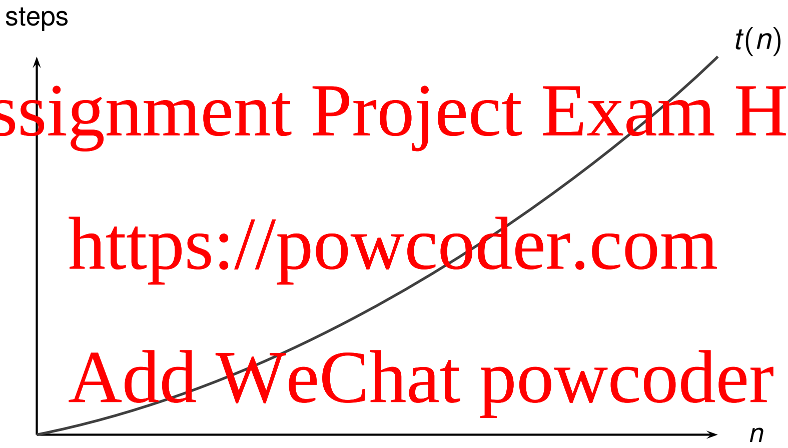
Suppose this is a plot of the running time $t(n)$ of our algorithm

Asymptotic Upper Bound: Example



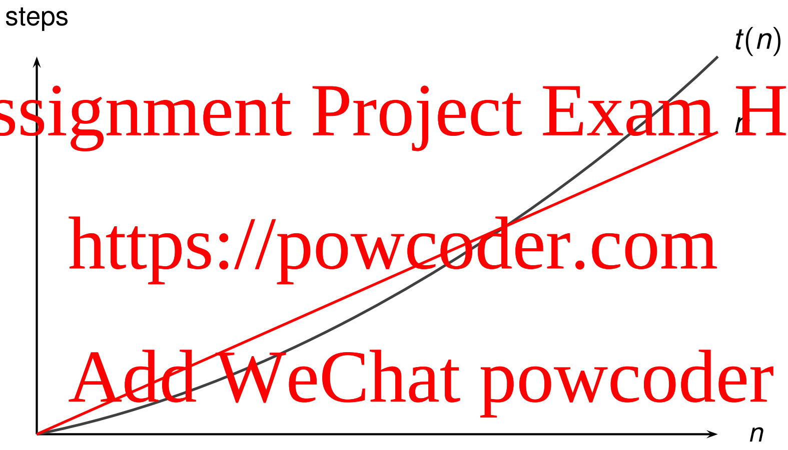
Can we find a function that will grow faster than this one?

Asymptotic Upper Bound: Example



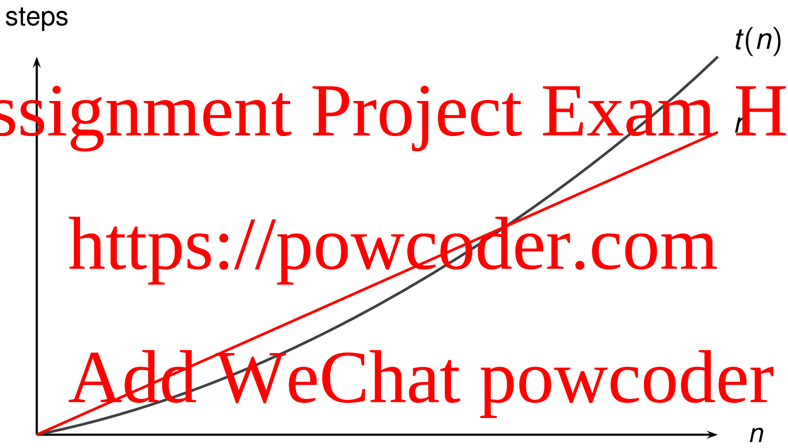
Let's consider a linear function: e.g. $f(n) = n$

Asymptotic Upper Bound: Example



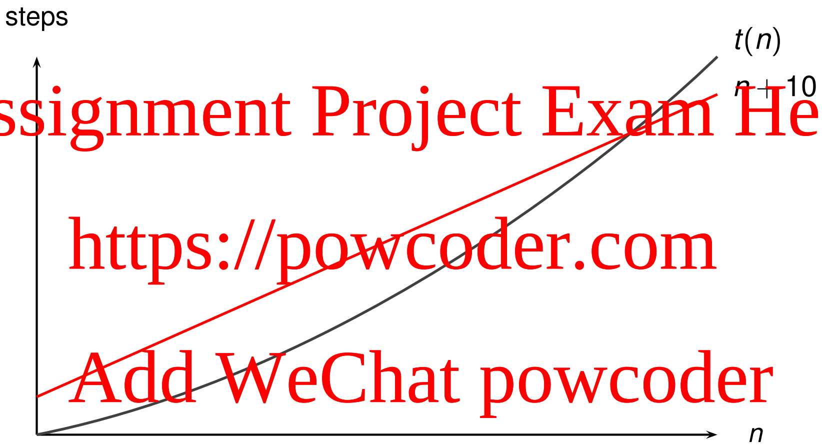
$f(n) = n$ isn't growing as fast as $t(n)$

Asymptotic Upper Bound: Example



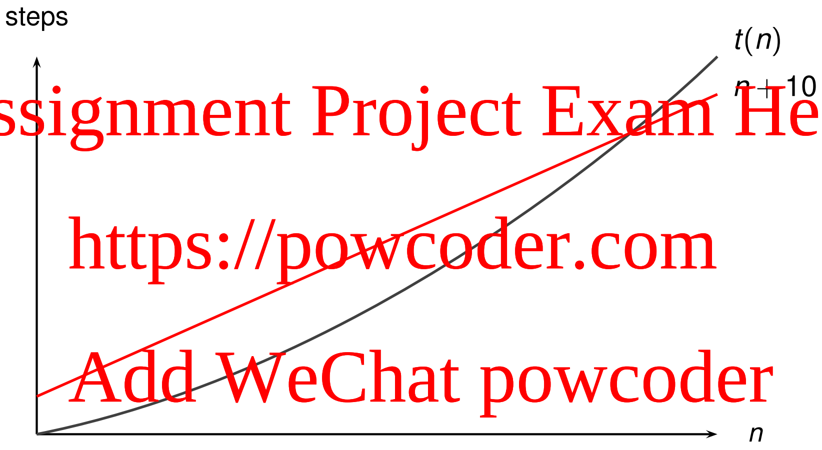
Try adding a constant: $f(n) = n + 10$

Asymptotic Upper Bound: Example



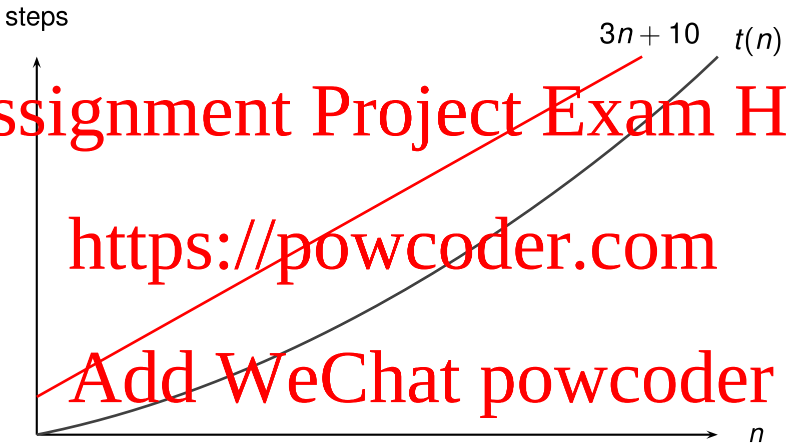
$f(n) = n + 10$ has the same rate of growth as n

Asymptotic Upper Bound: Example



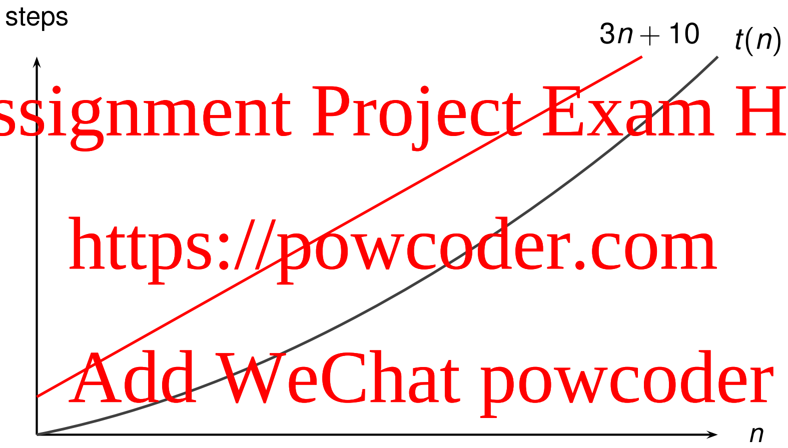
Try multiplying n by a constant $f(n) = 3n + 10$

Asymptotic Upper Bound: Example



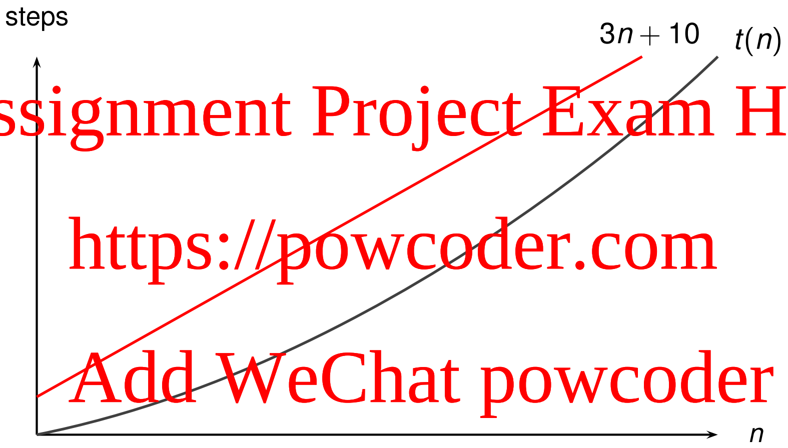
$f(n) = 3n + 10$ still doesn't grow as fast as $t(n)$

Asymptotic Upper Bound: Example



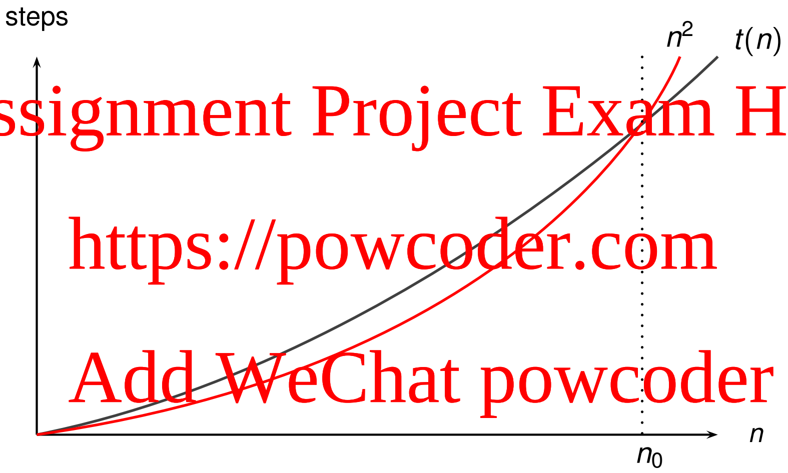
Any straight line (linear function) eventually gets overtaken by $t(n)$

Asymptotic Upper Bound: Example



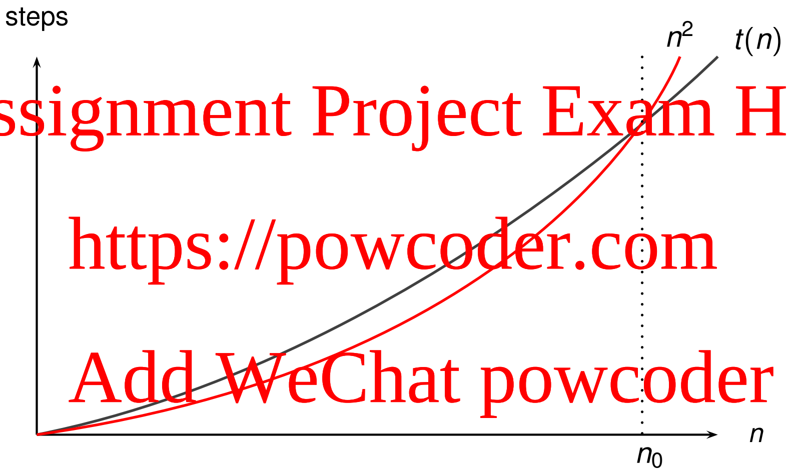
Need something that curves upwards: e.g. $f(n) = n^2$

Asymptotic Upper Bound: Example



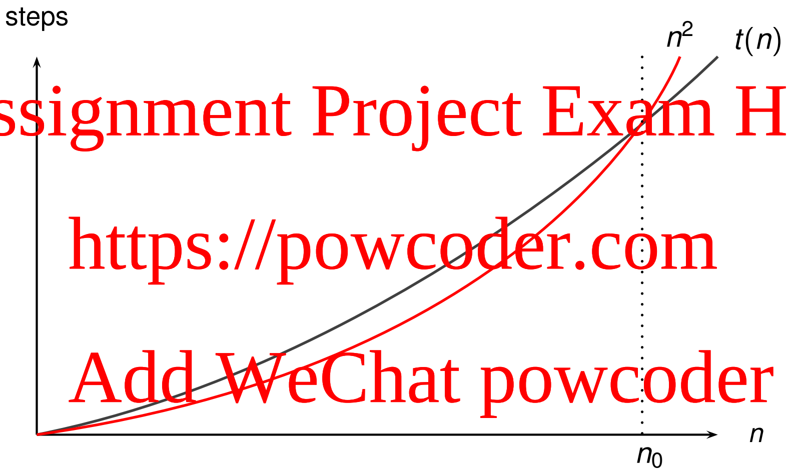
After crossing at n_0, n^2 remains above $t(n)$

Asymptotic Upper Bound: Example



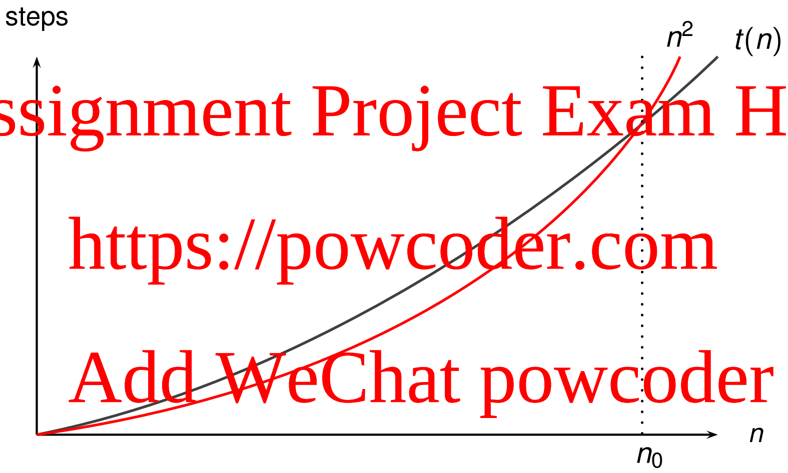
n^2 is an asymptotic upper bound for $t(n)$

Asymptotic Upper Bound: Example



so we say $t(n)$ is $O(n^2)$

Asymptotic Upper Bound: Example



Note that it doesn't matter that n^2 grows more rapidly than $t(n)$

Questions for you

Suppose $t(n) = n^2$

- What is the lowest value of n such that $t(n) \geq 10n + 20$?
- Note that n must be a whole number

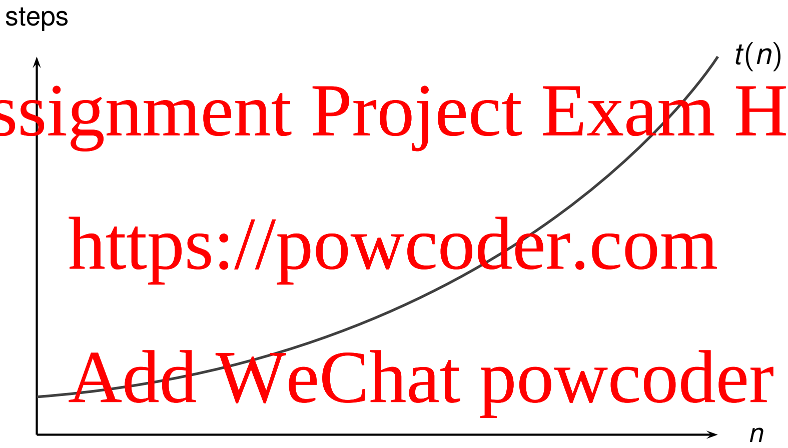
Questions for you

Suppose $t(n) = n^2$

- What is the lowest value of n such that $t(n) \geq 10n + 20$?
- Note that n must be a whole number

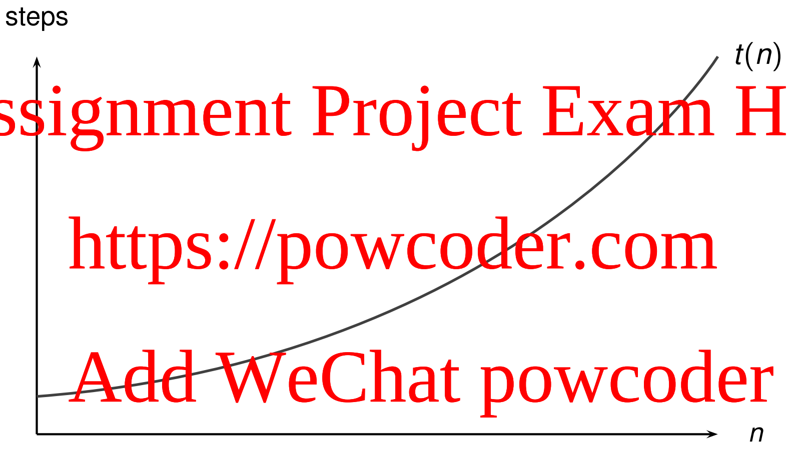
$n = 12$

Asymptotic Upper Bound: Another Example



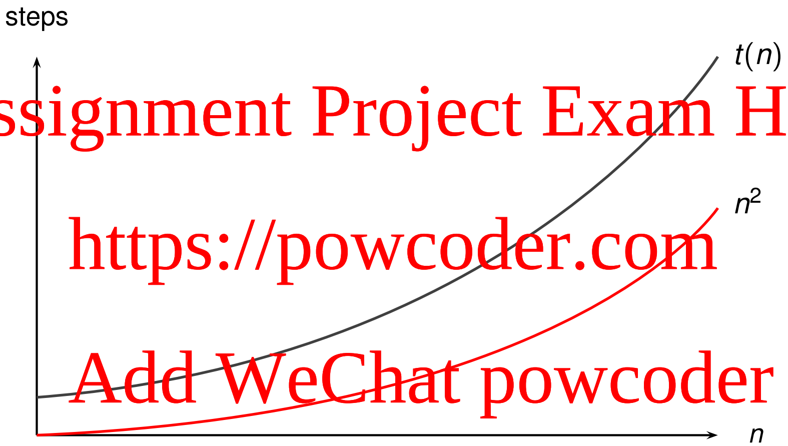
Consider another running time that grows more rapidly

Asymptotic Upper Bound: Another Example



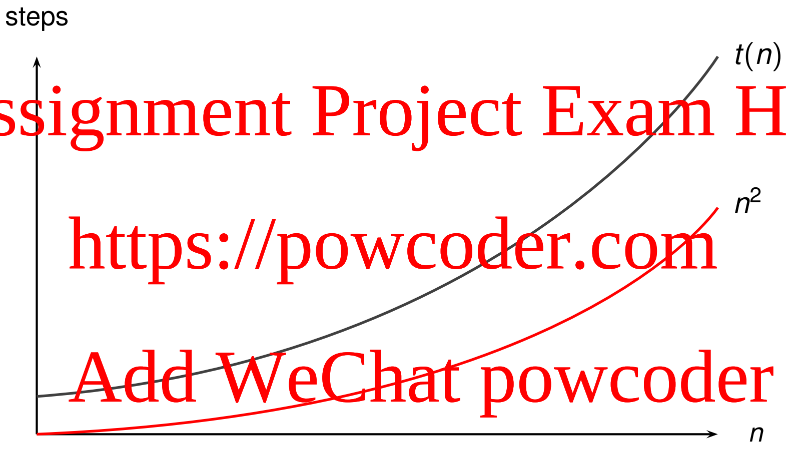
Try $f(n) = n^2$ as a possible asymptotic upper bound

Asymptotic Upper Bound: Another Example



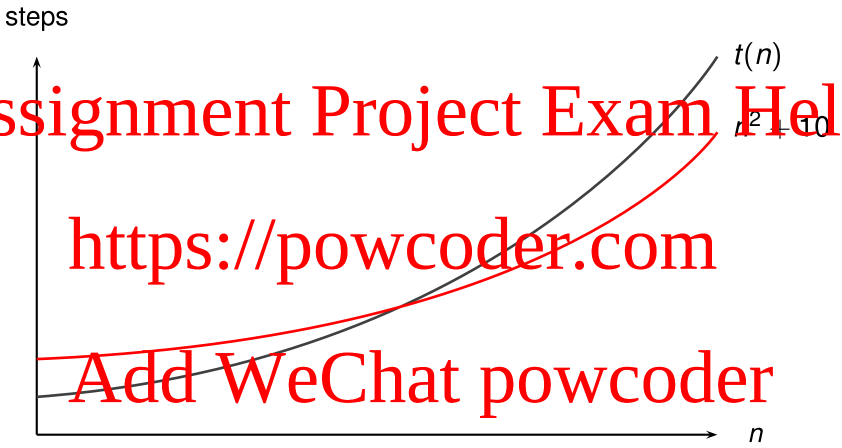
$f(n) = n^2$ doesn't seem to be catching up with $t(n)$

Asymptotic Upper Bound: Another Example



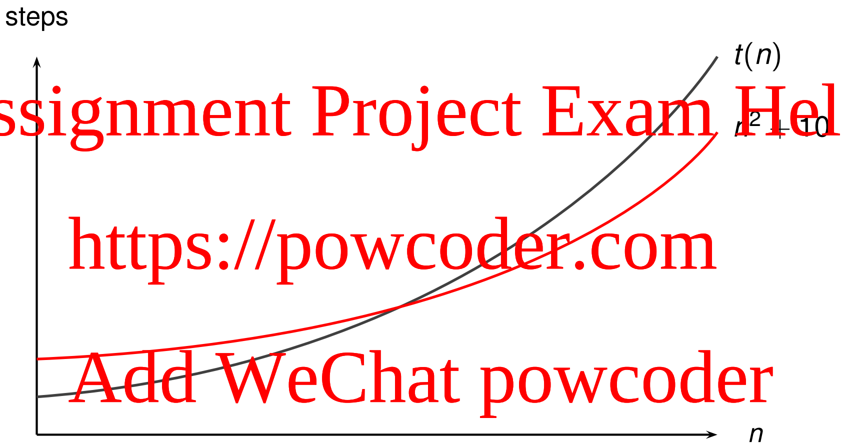
Try $f(n) = n^2 + 10$

Asymptotic Upper Bound: Another Example



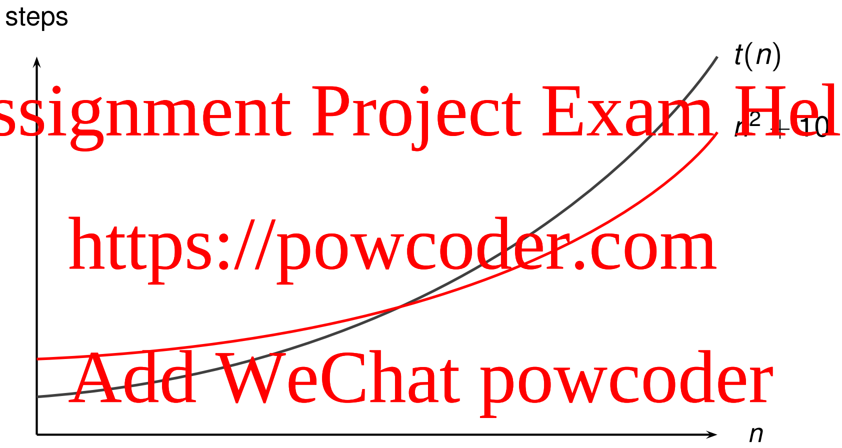
$f(n) = n^2 + 10$ is still not increasing at the required rate

Asymptotic Upper Bound: Another Example



Need something that grows faster

Asymptotic Upper Bound: Another Example

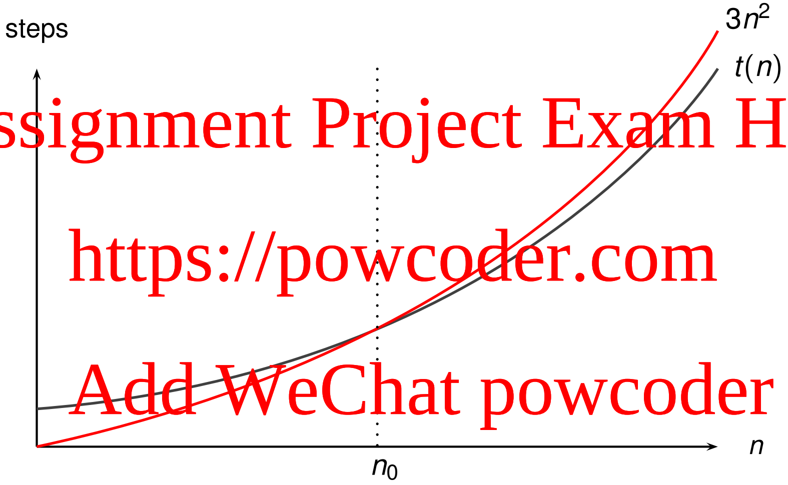


Try $f(n) = 3n^2$

Asymptotic Upper Bound: Another Example

$f(n) = 3n^2$ has adequate growth rate

Asymptotic Upper Bound: Another Example



After crossing at n_0, $3n^2$ stays ahead

Asymptotic Upper Bound: Another Example

$3n^2$ is an asymptotic upper bound for $t(n)$

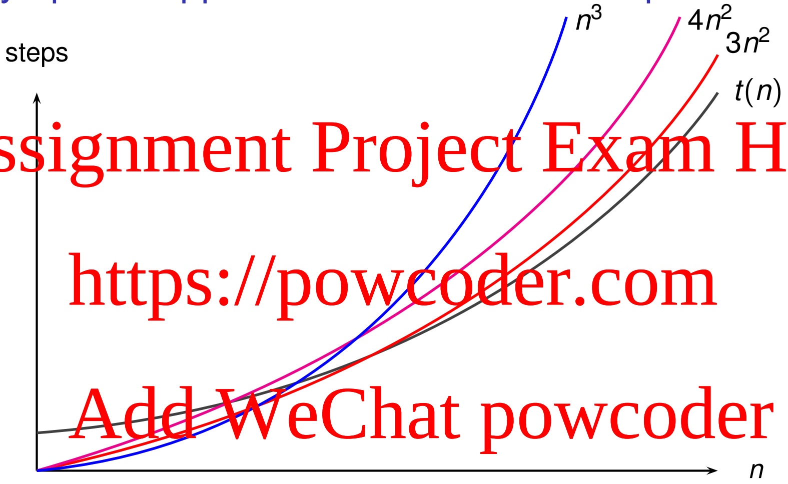
Asymptotic Upper Bound: Another Example

We say $t(n)$ is $O(n^2)$ (the constant can be dropped)

Asymptotic Upper Bound: Another Example

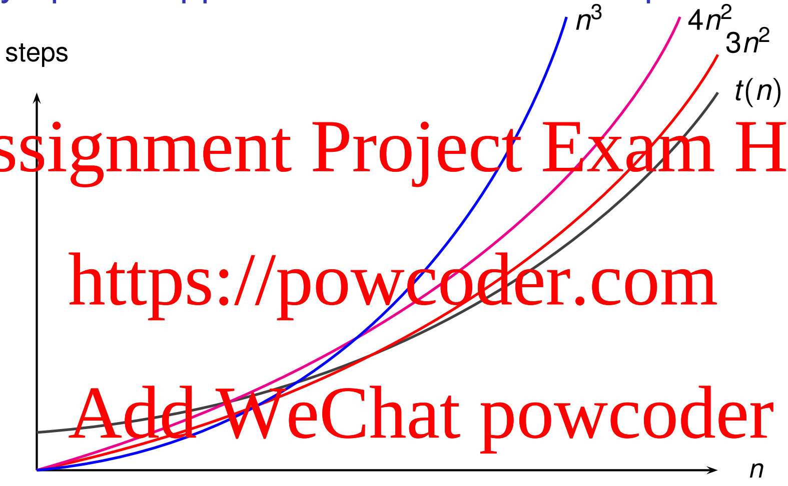
Note that $4n^2$ is also an asymptotic upper bound for $t(n)$

Asymptotic Upper Bound: Another Example



Even n^3 counts as an asymptotic upper bound for $t(n)$

Asymptotic Upper Bound: Another Example



So $t(n)$ is also $O(n^3)$

Asymptotic Upper Bound

Definition of $O(f(n))$:

$t(n)$ is $O(f(n))$ if

there are constants n_0 and $c > 0$ such that

$$t(n) \leq c \cdot f(n) \quad \textit{for all } n \geq n_0$$

Question for you

- Suppose that $t(n) = 3n^2 + 2n + 5$
- Give values for n_0 and c such that

$$t(n) \leq c \cdot n^2 \quad \text{for all } n \geq n_0$$

- Give whole number values that are as low as possible

Question for you

- Suppose that $t(n) = 3n^2 + 2n + 5$
- Give values for n_0 and c such that

$$t(n) \leq c \cdot n^2 \quad \text{for all } n \geq n_0$$

- Give whole number values that are as low as possible

$n_0 = 4$ and $c = 4$

Less Significant Terms

Consider the running time:

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

Less Significant Terms

Consider the running time:

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

Compare this to the function

$$f(n) = 15 \cdot n^4$$

Less Significant Terms

Consider the running time:

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

Compare this to the function

$$f(n) = 15 \cdot n^4$$

$f(n)$ will *eventually* catch up with $t(n)$

Less Significant Terms

Consider the running time:

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

Compare this to the function

$$f(n) = 15 \cdot n^4$$

$f(n)$ will *eventually* catch up with $t(n)$

So only consider highest order (most significant) term

Less Significant Terms

Consider the running time:

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

Compare this to the function

$$f(n) = 15 \cdot n^4$$

$f(n)$ will *eventually* catch up with $t(n)$

So only consider highest order (most significant) term

$t(n)$ is $O(n^4)$

Question for you

What is the lowest value of n where $f(n) = 15 \cdot n^4$ is greater than

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

Question for you

What is the lowest value of n where $f(n) = 15 \cdot n^4$ is greater than

$$t(n) = 14n^4 + 12n^3 \log n + 47n^2 + 3n + 17$$

$n = 76$

Ignoring Constants

Consider the running time

$$t(n) = 100000n^4$$

Ignoring Constants

Consider the running time

$$t(n) = 100000n^4$$

$t(n)$ is $O(n^4)$

Ignoring Constants

Consider the running time

$$t(n) = 100000n^4$$

$t(n)$ is $O(n^4)$

Consider the rather different running time

$$t(n) = 0.000001n^4$$

Ignoring Constants

Consider the running time

$$t(n) = 100000n^4$$

$t(n)$ is $O(n^4)$

Consider the rather different running time

$$t(n) = 0.000001n^4$$

$t(n)$ is still $O(n^4)$

Ignoring Constants

Consider the running time

$$t(n) = 100000n^4$$

$t(n)$ is $O(n^4)$

Consider the rather different running time

$$t(n) = 0.000001n^4$$

$t(n)$ is still $O(n^4)$

Lack of concern for constants is consistent with our lack of concern for the granularity with which we measure running time

Landmark Growth Rates

We now consider a few of the more common growth rates

Constant Time

$f(n) = 1$

- Expression of upper bound, written $O(1)$
- Same as $O(2)$, $O(3)$, ...
- Referred to as **constant time**
- Running time doesn't increase with size of problem

Example: Operations on a stack (push, pop, etc)

Linear Time

$T(n) = n$

- Expression of upper bound, written $O(n)$
- Referred to as **linear time**
- Doubling problem size doubles running time

Example: Linear search: finding largest element in list

Polynomial Time

$f(n) = n^k$ for some $k \geq 1$

- Expression of upper bound, written $O(n^k)$
- Referred to as **polynomial time**
- k is a positive integer
- e.g. $O(n)$, $O(n^2)$, $O(n^3)$, $O(n^4)$, ...
- Linear time special case of polynomial time ($k = 1$)
- $O(n^2)$ is called **quadratic**
- $O(n^3)$ is called **cubic**

Example: $O(n^2)$ - selection sort; insertion sort

Exponential Time

$f(n) = k^n$ for some $k > 1$

- Expression of upper bound, written $O(k^n)$
- Referrred to as **exponential time**
- Typically $k = 2$, so $O(2^n)$
- Unacceptable running time

Example: Any algorithm that considers all subsets of a set

Logarithmic Time

$f(n) = \log n$

- Expression of upper bound, written $O(\log n)$
- Referred to as **logarithmic time**
- Grows very slowly
- Base of logarithm not important when considering $O(.)$
- $O(\log_2 n)$ same as $O(\log_3 n)$, ...

Example: binary search

Another Common Running Time

$f(n) = n \log n$

- Expression upper bound, written $O(n \log n)$
- Grows only slightly faster than $O(n)$

Example: merge sort

Relative Growth Rates

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < \ldots < O(2^n)$$

Importance of Running Times

problem size	running time					
	n	$n \log n$	n^2	n^3	2^n	$n!$
10	<1 s	<1 s	<1 s	<1 s	<1 s	4 s
30	<1 s	<1 s	<1 s	<1 s	18 m	10^{25} y
50	<1 s	<1 s	<1 s	<1 s	36 y	$>10^{25}$ y
100	<1 s	<1 s	<1 s	1 s	10^{17} y	$>10^{25}$ y
10^3	<1 s	<1 s	1 s	18 m	$>10^{25}$ y	$>10^{25}$ y
10^4	<1 s	<1 s	2 m	12 d	$>10^{25}$ y	$>10^{25}$ y
10^5	<1 s	2 s	3 h	32 y	$>10^{25}$ y	$>10^{25}$ y
10^6	1 s	20 s	12 d	31,710 y	$>10^{25}$ y	$>10^{25}$ y

assumes 1,000,000 instructions per second

Questions for you

Suppose we are told that an algorithm has a running time of $O(n^3)$

Question: What does this tell us about how *slow* the running time is?

Question: What does this tell us about how *fast* the running time is?

Questions for you

Suppose we are told that an algorithm has a running time of $O(n^3)$

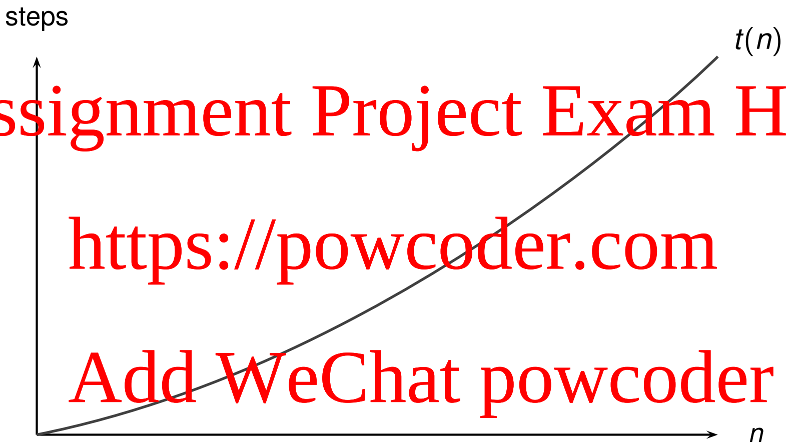
Question: What does this tell us about how *slow* the running time is?

The running time is asymptotically no worse than $c \cdot n^3$ for some c

Question: What does this tell us about how *fast* the running time is?

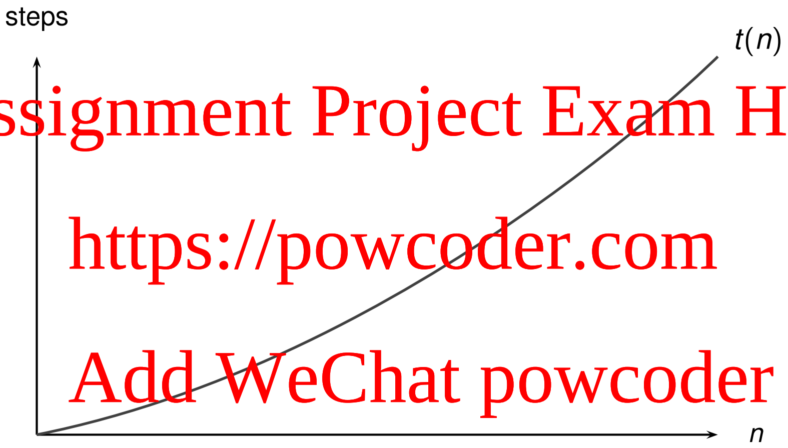
Nothing

Asymptotic Lower Bound: Example



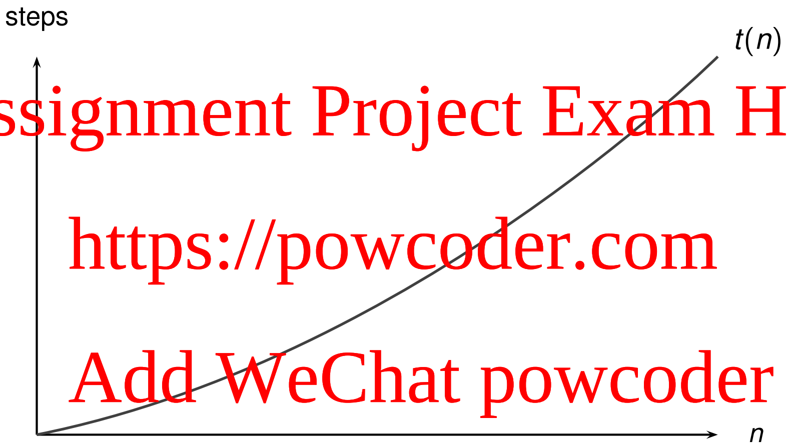
Suppose this is a plot of the running time $t(n)$ of our algorithm

Asymptotic Lower Bound: Example



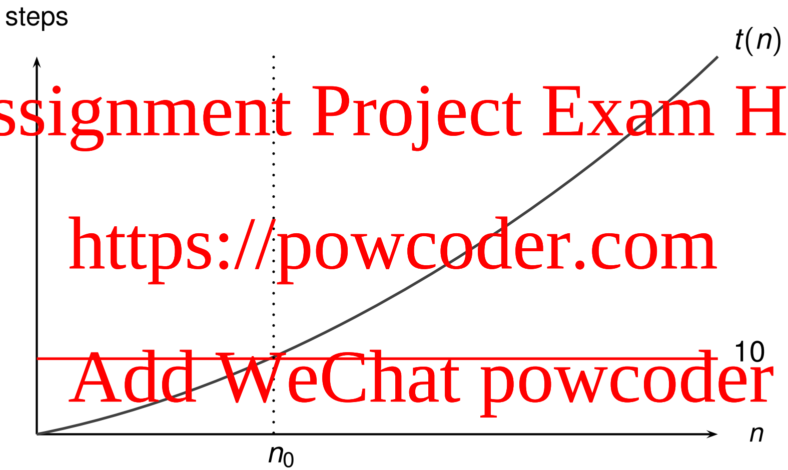
Can we find a function that will always keep *below* this one?

Asymptotic Lower Bound: Example



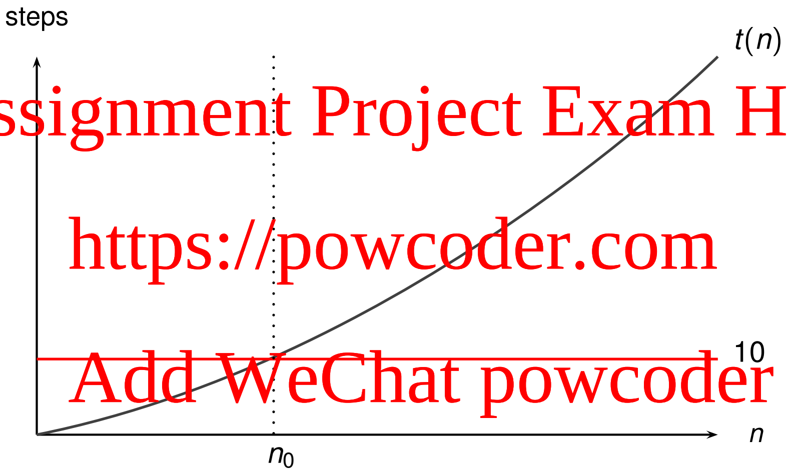
Let's consider a constant function: e.g. $f(n) = 10$

Asymptotic Lower Bound: Example



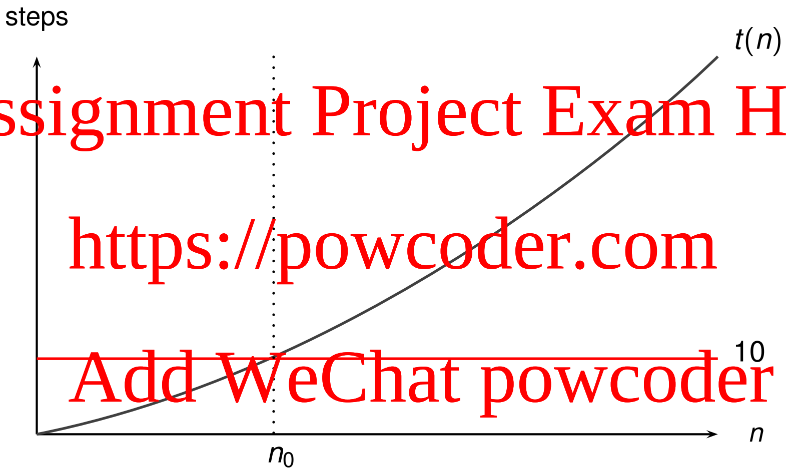
After crossing at n_0, 10 remains below $t(n)$

Asymptotic Lower Bound: Example



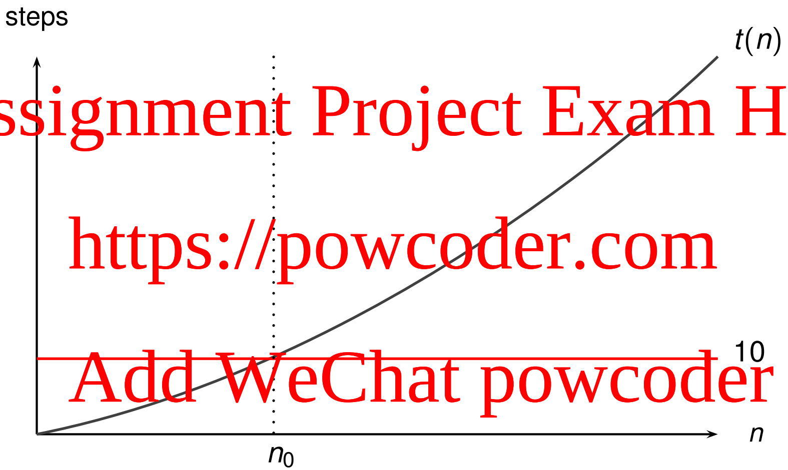
So we can say $t(n)$ is $\Omega(10)$

Asymptotic Lower Bound: Example



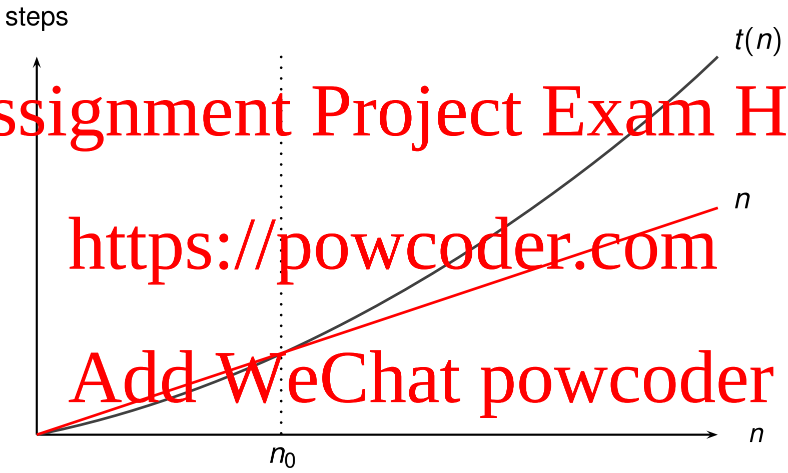
Prefer to say $t(n)$ is $\Omega(1)$

Asymptotic Lower Bound: Example



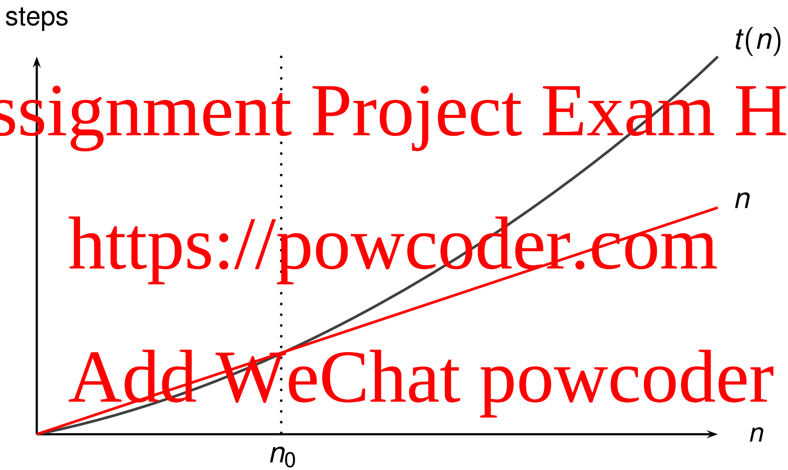
Let's try a function that grows faster: e.g. n

Asymptotic Lower Bound: Example



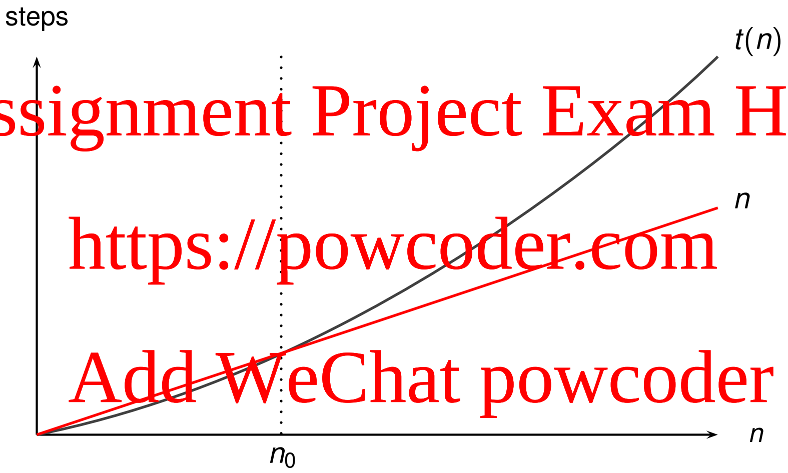
$f(n) = n$ lies below $t(n)$ after n_0

Asymptotic Lower Bound: Example



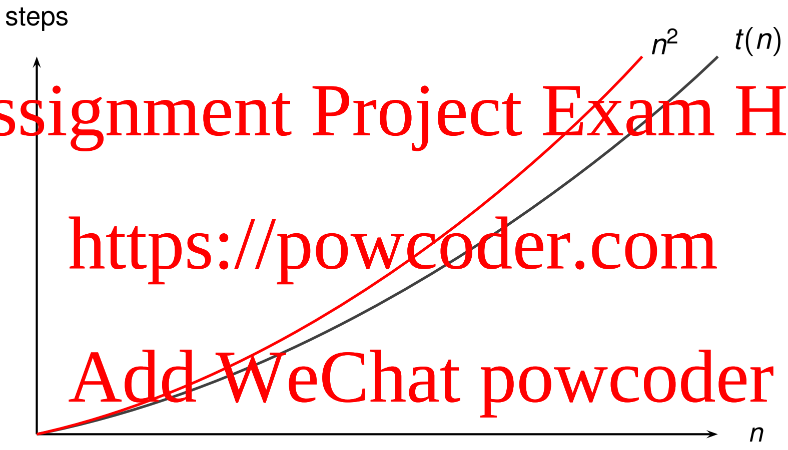
So we can also say $t(n)$ is $\Omega(n)$

Asymptotic Lower Bound: Example



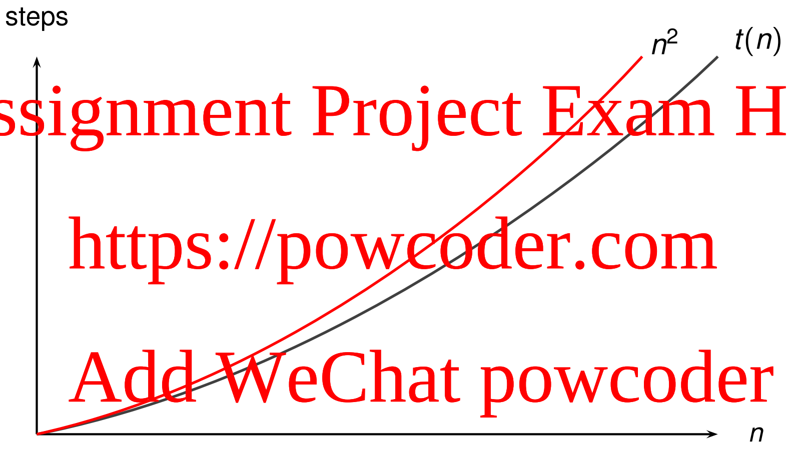
Let's try a function that grows faster still: e.g. n^2

Asymptotic Lower Bound: Example



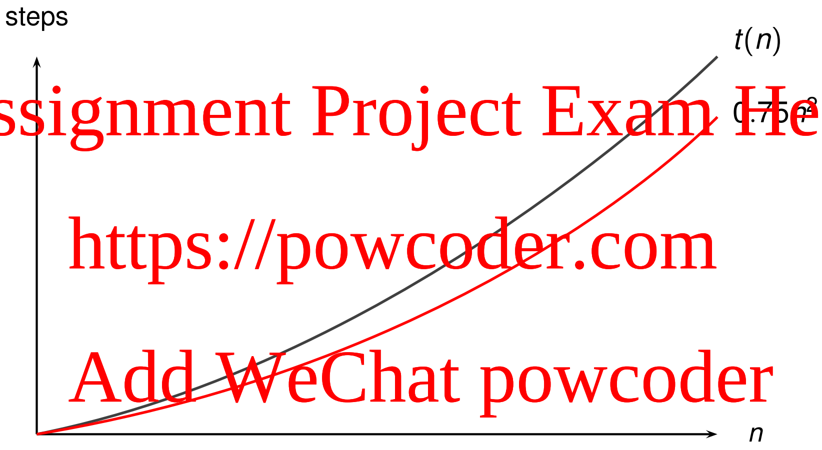
$f(n) = n^2$ grows faster than $t(n)$

Asymptotic Lower Bound: Example



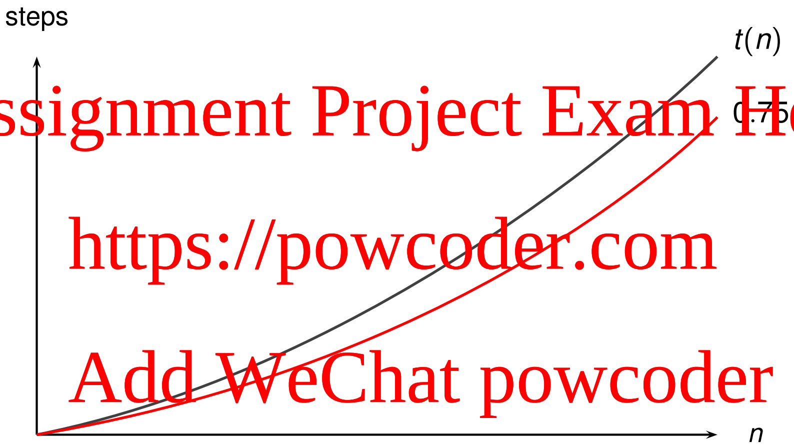
What about $f(n) = 0.75n^2$?

Asymptotic Lower Bound: Example



$f(n) = 0.75n^2$ grows slower than $t(n)$

Asymptotic Lower Bound: Example



So we can say that $t(n)$ is $\Omega(n^2)$

Asymptotic Lower Bound

Definition of $\Omega(f(n))$:

$t(n)$ is $\Omega(f(n))$ if

there are constants n_0 and $c > 0$ such that

$$t(n) \geq c \cdot f(n) \quad \textit{for all } n \geq n_0$$

Question for you

- Let $t(n) = n^2 - n$
- Give values for n_0 and c showing that $t(n)$ is $\Omega(n^2)$

$t(n)$ is $\Omega(f(n))$ if

there are constants n_0 and $c > 0$ such that

$$t(n) \geq c \cdot f(n) \text{ for all } n \geq n_0$$

Question for you

- Let $t(n) = n^2 - n$
- Give values for n_0 and c showing that $t(n)$ is $\Omega(n^2)$

$t(n)$ is $\Omega(f(n))$ if

there are constants n_0 and $c > 0$ such that

$$t(n) \geq c \cdot f(n) \quad \textit{for all } n \geq n_0$$

$n_0 = 4$ and $c = 0.75$

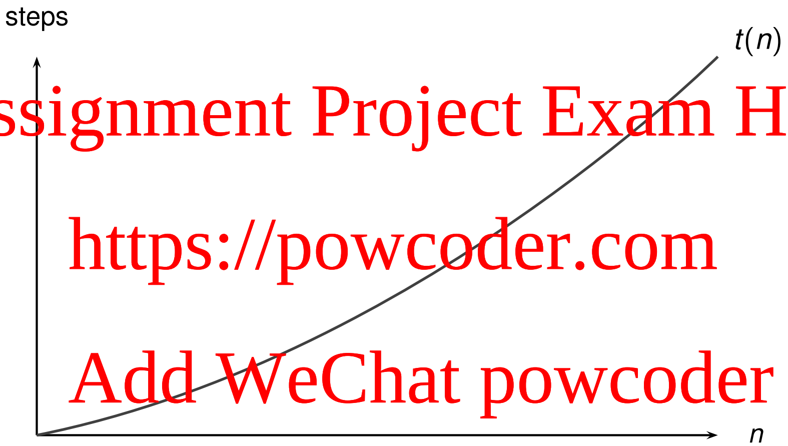
Asymptotic Tight Bound

Definition of $\Theta(f(n))$:

$t(n)$ is $\Theta(f(n))$ if

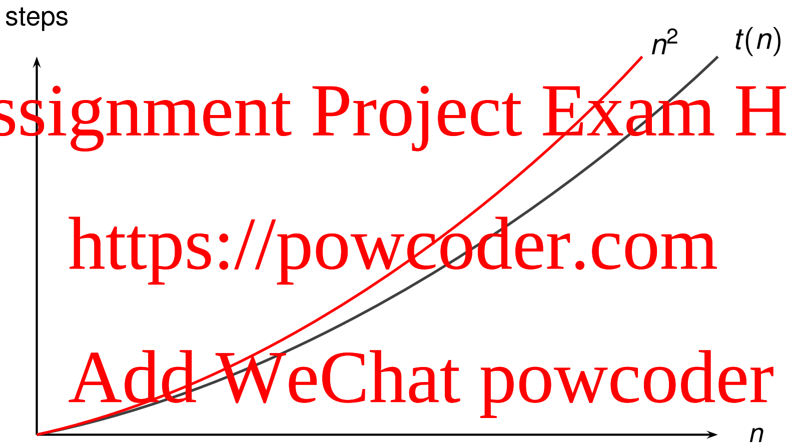
$t(n)$ is $O(f(n))$ and $t(n)$ is also $\Omega(f(n))$

Asymptotic Tight Bound: Example



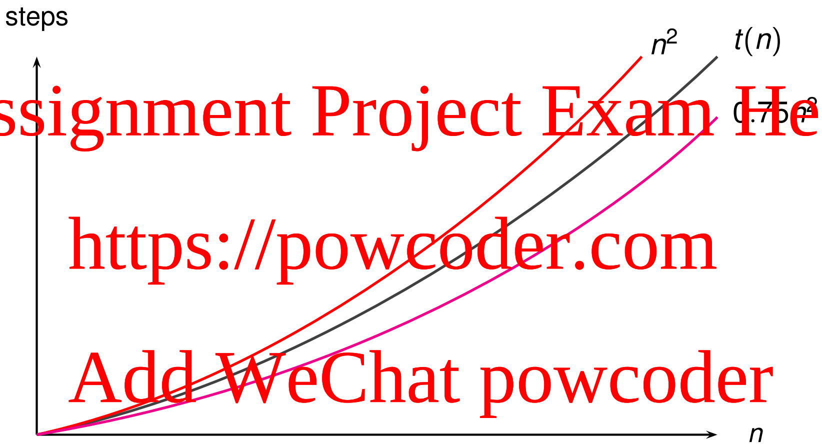
Same function as previous example

Asymptotic Tight Bound: Example



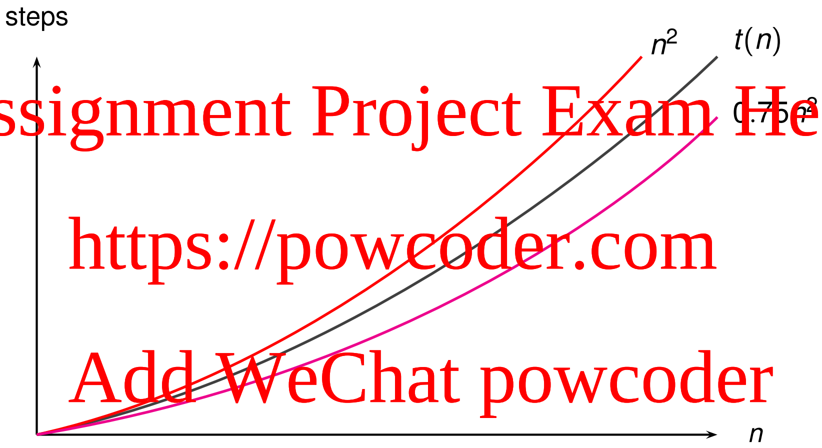
$f(n) = n^2$ grows faster than $t(n)$

Asymptotic Tight Bound: Example



$f(n) = 0.75n^2$ grows slower than $t(n)$

Asymptotic Tight Bound: Example



So we can say that $t(n)$ is $\Theta(n^2)$

Have we been oversimplifying things?

Question: Is running time really *just* a function of n?

Have we been oversimplifying things?

Question: Is running time really *just* a function of n?

Answer: Sometimes, but not always

Have we been oversimplifying things?

Question: Is running time really *just* a function of n?

Answer: Sometimes, but not always

Question: What else can running time be a function of?

Have we been oversimplifying things?

Question: Is running time really *just* a function of n?

Answer: Sometimes, but not always

Question: What else can running time be a function of?

Answer: Sometimes the *values* make a difference

Example: The Cases of Searching

Consider algorithm that searches an unordered list for a target value by considering each item in list in turn from first to last

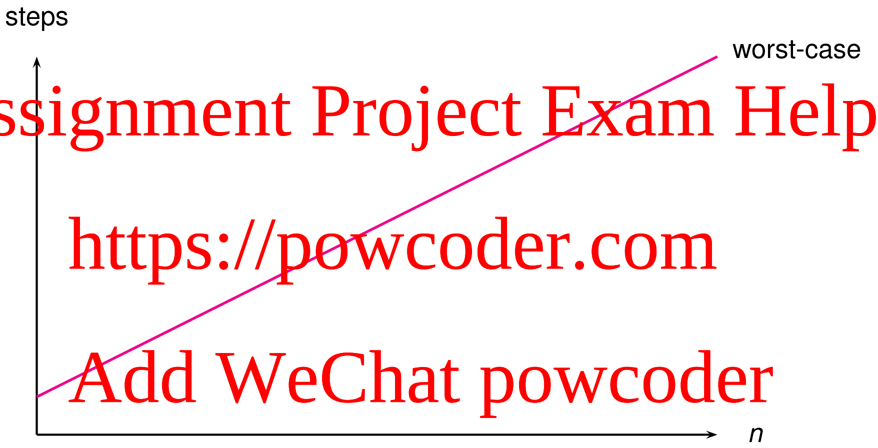
Best-case:

- inputs where target value at front of list
- running time will be $\Theta(1)$

Worst-case:

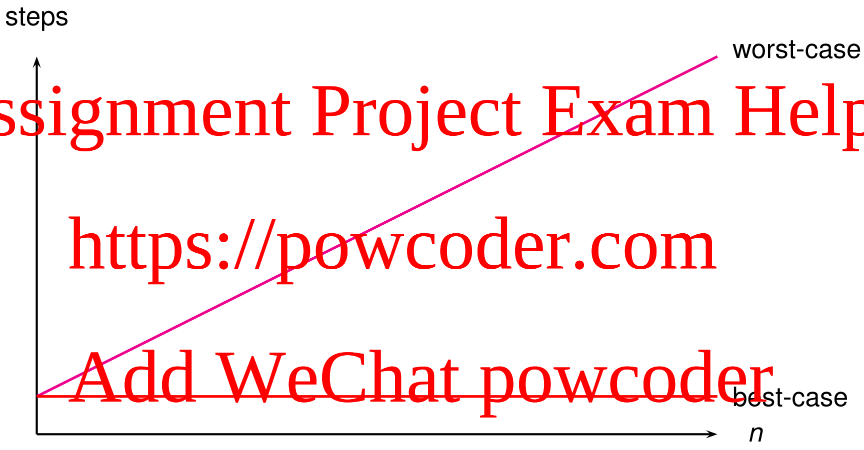
- inputs where target value not in list
- running time will be $\Theta(n)$ where n is length of list

Worst- Best-Case Difference



Worst-case of $\Theta(n)$

Worst- Best-Case Difference



Best-case of $\Theta(1)$

Expected- or Average-Case

Expected performance: the running time you would expect on average

Something that is of interest when worst-case and best-case running times are asymptotically different

Difficulty: over what selection of inputs should the calculation take place?

- Values making up input (not just its size) matters
- Probabilities of each possible input might not form a uniform distribution
- Actual distribution once deployed may be unknowable

Expected- or Average-Case: Example

Consider linear search among n items

Some of the issues that arise:

- Need to know size of set of all the items that could appear in list
- Can we assume that items in list chosen at random from this set
- Are items selected from this set with or without replacement

Resolving these issues would allow us to:

- Estimate the probability that target item is in a list of size n
- Estimate the expected number of comparisons

Lower Bound for a Problem

A different kind of question:

what is the lower limit on how efficiently a problem can be solved?

- No particular algorithm in mind
- This is a property of a **problem** not of an **algorithm**
- This can be a very hard question to answer
- Requires generalising across all possible algorithms

An Example: Sorting Problem

Question:

What is the lower bound for the sorting problem?

- We know we can sort in $O(n \log n)$ time

  — c.f. MergeSort

- Can we sort any quicker?
- Is it possible that there an algorithm that solves the sorting problem that has a worst-case running time that is better than $O(n \log n)$?

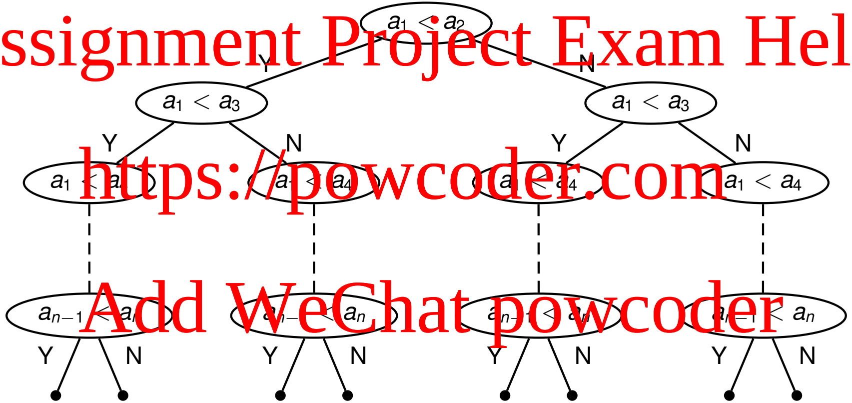
Consider Comparison Sorting Algorithms only

Comparison sorting algorithms sort elements by comparing them with each other

- Bucket sort is **not** a comparison sorting algorithm
- Fixed set of buckets into which elements placed
- Buckets hold all elements in a certain range
- No comparison between elements

Comparison Sort Decision Tree

Sorting elements $(a_1, \ldots, a_n)$



There are $n!$ leaves — one for each permutation

Implications of Decision Tree

The height of the decision tree gives lower bound on length of computations

- There must be a distinct computation path for each root to leaf path in the tree

  — otherwise algorithm is not correct

- How long must an algorithm's computations be if it is capable of distinguishing so many alternatives?

Analysis of Decision Tree

- Decision tree has $n!$ leaves
- Height is at least $\log n!$
- $n!$ has at least $\frac{n}{2}$ terms that are at least $\frac{n}{2}$
- So $\log n! \geq \log\left(\frac{n}{2}^{\frac{n}{2}}\right)$
- $\log(\frac{n}{2}^{\frac{n}{2}}) = \frac{n}{2} \log \frac{n}{2}$
- $\frac{n}{2} \log \frac{n}{2}$ is $O(n \log n)$
- Lower bound for comparison-based sorting is $O(n \log n)$

Question for you

- Why is it not strictly correct to start a sentence as follows.
  - The asymptotic upper bound for the running time of the algorithm is …

Question for you

- Why is it not strictly correct to start a sentence as follows

  - The asymptotic upper bound for the running time of the algorithm is …

There can never be just one asymptotic upper bound, so the definite article “the” is not appropriate.

- An asymptotic upper bound for the running time of the algorithm is …

Question for you

- Does it make sense to specify either of the following? Explain your answer.

  - An asymptotic lower bound on the worst-case running time of an algorithm;
  - An asymptotic upper bound on the best-case running time of an algorithm.

Question for you

- Does it make sense to specify either of the following? Explain your answer.

  - An asymptotic lower bound on the worst-case running time of an algorithm;
  - An asymptotic upper bound on the best-case running time of an algorithm.

Yes. An asymptotic lower bound on the worst-case running time gives a constraint on how fast the algorithm could run when exhibiting its worst-case behaviour, and an asymptotic upper bound on the best-case running time gives a constraint on how slow the algorithm could run when exhibiting its best-case behaviour

Question for you

- Give an example of an algorithm for which an asympotic lower bound for the worst-case running time is not also an asympotic lower bound for the running time in general.

Question for you

- Give an example of an algorithm for which an asympotic lower bound for the worst-case running time is not also an asympotic lower bound for the running time in general.

An algorithm that performs linear left-to-right search of a sequence of length n for a value. An asymptotic lower bound on worst-case is $\Omega(n)$, but this is not a lower bound for the algorithm in general.

Question for you

- Are upper and lower bounds for the running time of an algorithm also upper and lower bounds for the worst-case running time of an algorithm and upper and lower bounds for the best-case running time of an algorithm?

Question for you

- Are upper and lower bounds for the running time of an algorithm also upper and lower bounds for the worst-case running time of an algorithm and upper and lower bounds for the best-case running time of an algorithm?

Yes, though they may not be as tight as the bounds that could be given for the worst- or best-case running times.