http://algs4.cs.princeton.edu

1.4 ANALYSIS OF ALGORITHMS

- *complexity of loops*
- *exponentials and logarithms*
- *order-of-growth classifications*
- dependencies on input
- memory
- summary of module 1

http://algs4.cs.princeton.edu

1.4 ANALYSIS OF ALGORITHMS

- *complexity of loops*
- *exponentials and logarithms*
- *order-of-growth classifications*
- dependencies on input
- memory
- summary of module 1

Bounding complexity of loops

Consecutive loops: Add the time complexity

```
int count = 0;
for (int i = 0; i < n; i++)
   count++;
for (int j = 0; j < m; j++)
   count++;
```

Bounding complexity of loops

Simple nested loops : Product of the loop numbers

n=2 m=4

```
int count = 0;
for (int i = 0; i < n; i++)
   for (int j = 0; j < m; j++)
      count++;
```

i,j
0,0
0,1
0,2
0,3
1,0
1,1
1,2
1,3

Interrelated nested loops: Sum of the inner loop numbers

n=4

```
int count = 0;
for (int i = 0; i < n; i++)
   for (int j = 0; j < i*i; j++)
      count++;
```

i,j
0,-
1,0
2,0 2,1 2,2 2,3
3,0 3,1 3,2 3,3 3,4
3,5 3,6 3,7 3,8

Talning umferða

A.

```
sum = 0;
for (int i=0; i < n; i++)
 for (int j=0; j < n; j++)
     for (int k=0; k < n; k++)
        sum = sum + i*j*k;
```

Svar: $T(n) = n^3$

B.

```
sum = 0;
for (int i=0; i < n; i++)
 for (int j=0; j < 10; j++)
     sum = sum + i*j;
```

Svar: $T(n) = 10\ n$

"How many iterations?"

```
// Returns largest element of a lower triangular matrix
int maxElt(int A[][], int n) {
   int max= A[0][0];
   for (int i=0; i < n; i++)
       for (int j=0; j <= i; j++)
           if (A[i][j] > max)    max = A[i][j];
   return max;
}
```

The number of iterations of inner loop is: $$1+2+3+\ldots+n=\sum_{i=1}^{n} i=?$$

Evaluating sums

$$1+2+3+...+16=\sum_{i=1}^{16} i$$

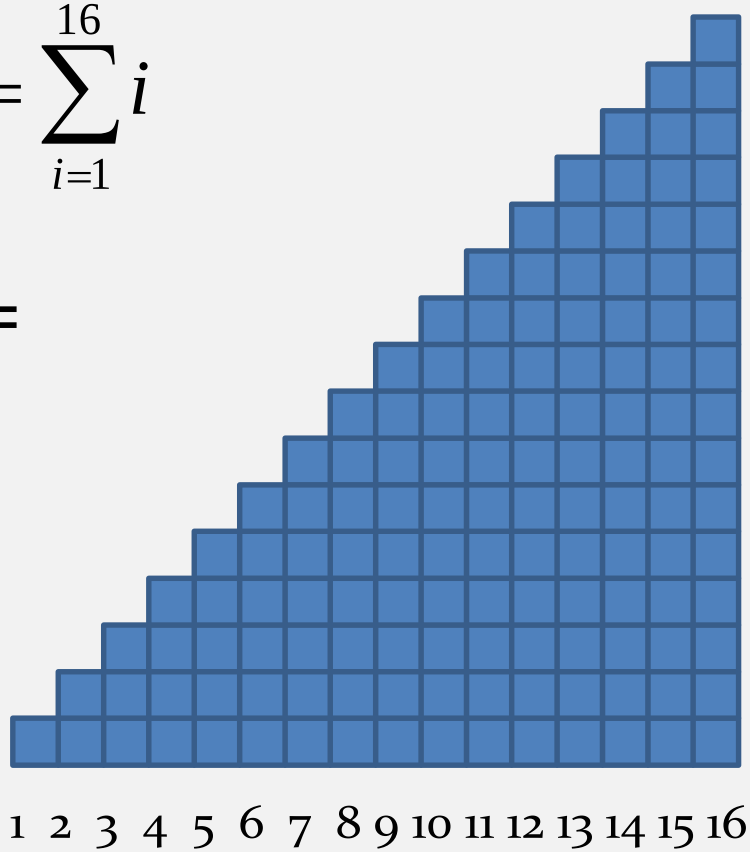
$$=$$

7

Evaluating sums

$$1+2+3+\ldots+16=\sum_{i=1}^{16} i$$

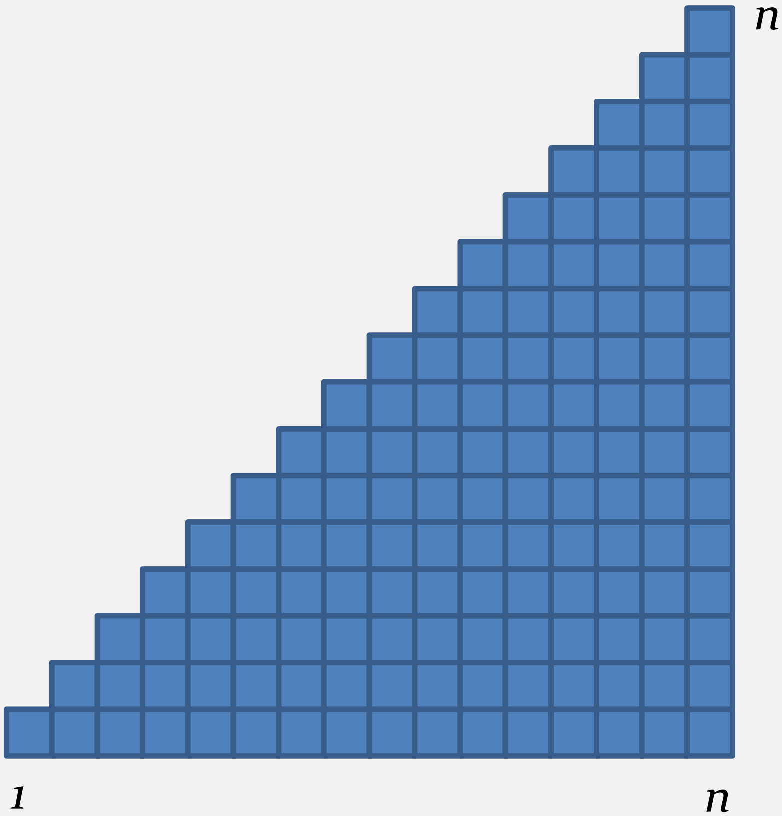
$$=$$



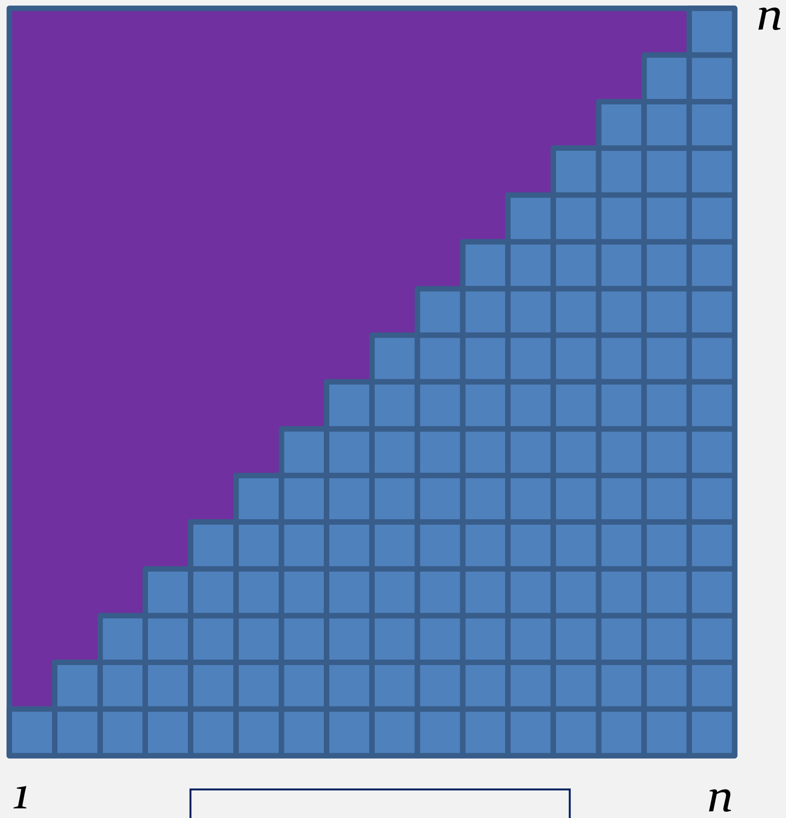
8

Bounding sums

$$1+2+3+\ldots+n=\sum_{i=1}^{n} i$$



$\leq$

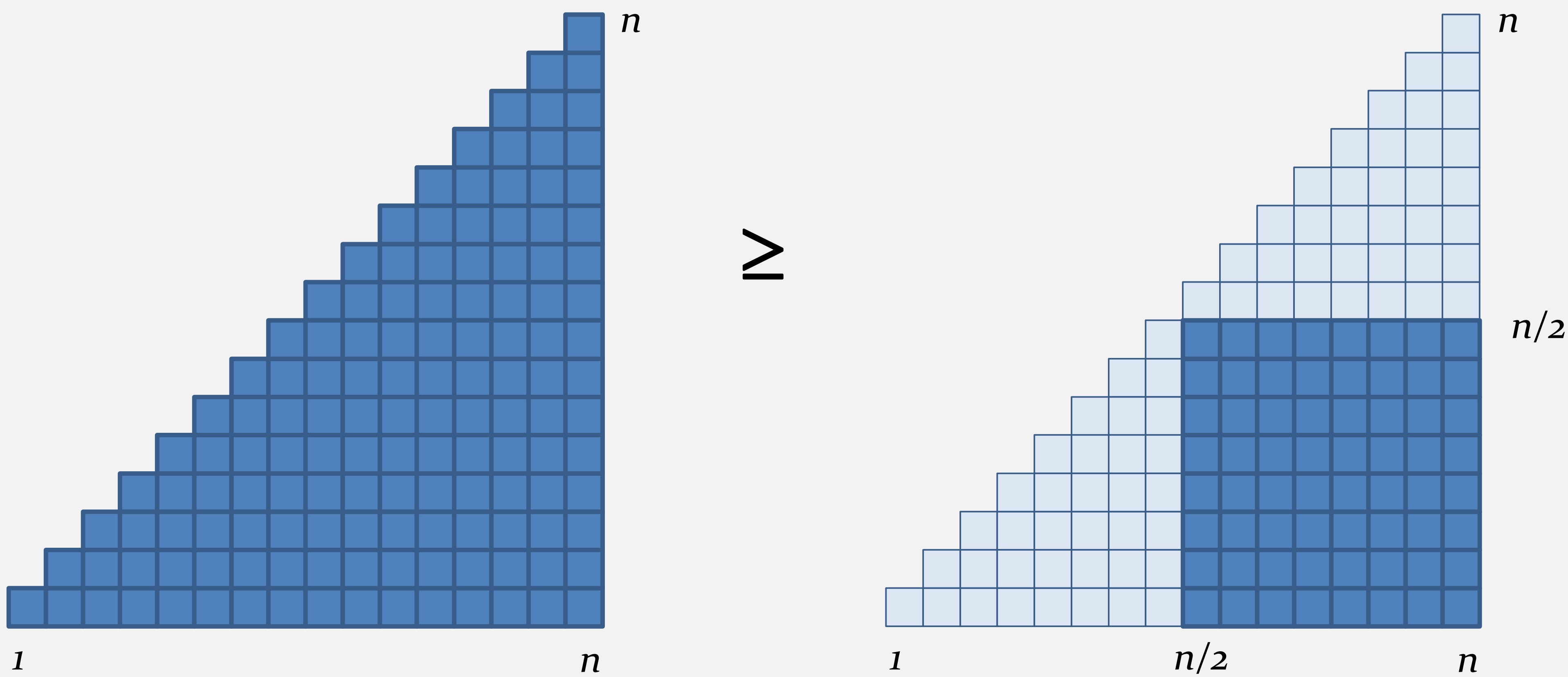


$$\boxed{n \cdot n = n^2}$$

9

Bounding sums

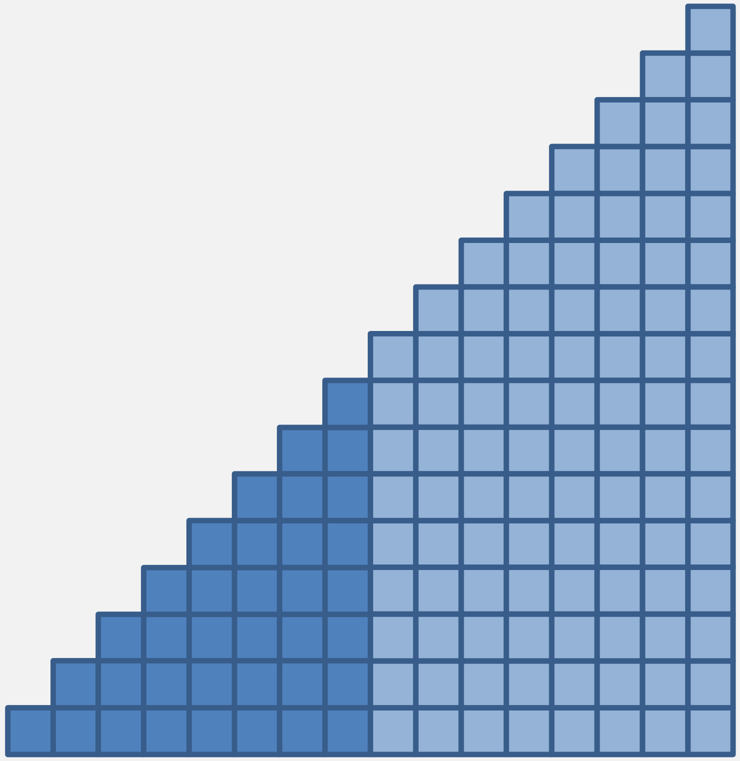
$$1 + 2 + 3 + \ldots + n = \sum_{i=1}^{n} i$$



$$\boxed{n/2 \cdot n/2 = n^2/4}$$

Bounding sums

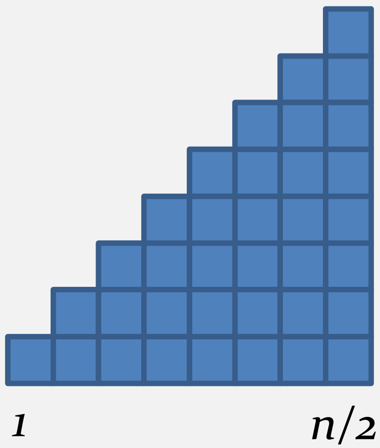
$$1+2+3+\ldots+n=\sum_{i=1}^{n} i$$

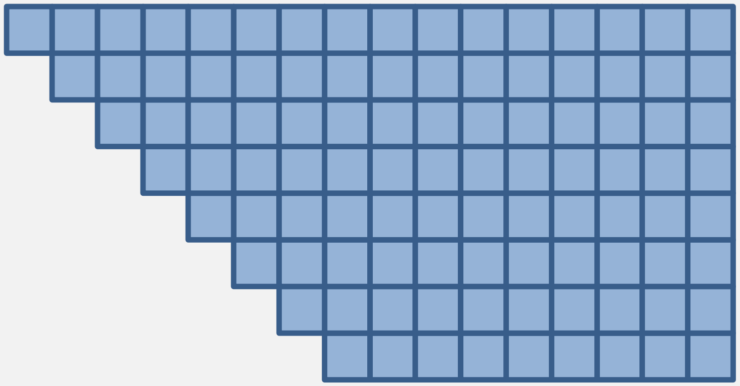


11

Bounding sums

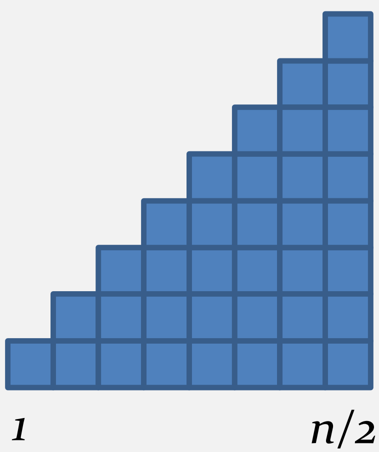
$$1+2+3+\ldots+n=\sum_{i=1}^{n} i$$

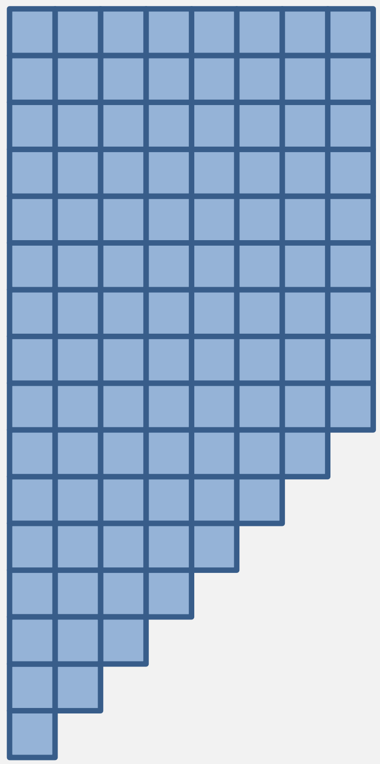




Bounding sums

$$1 + 2 + 3 + \ldots + n = \sum_{i=1}^{n} i$$

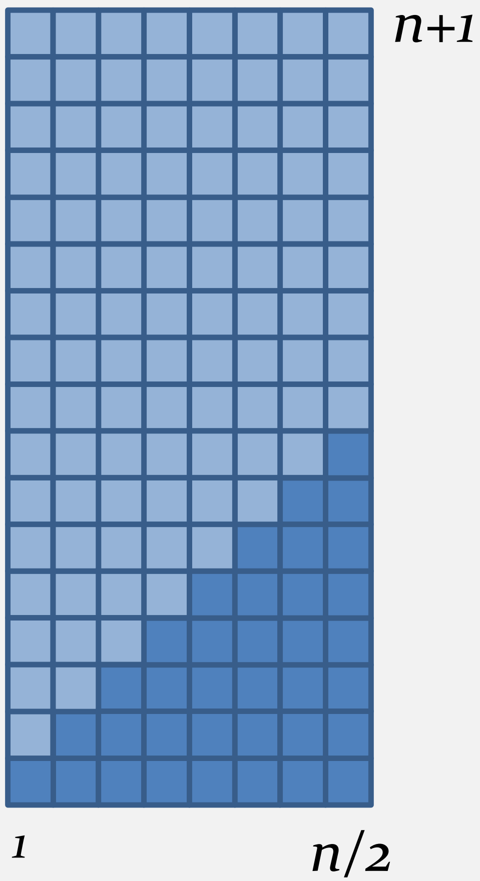




13

Bounding sums

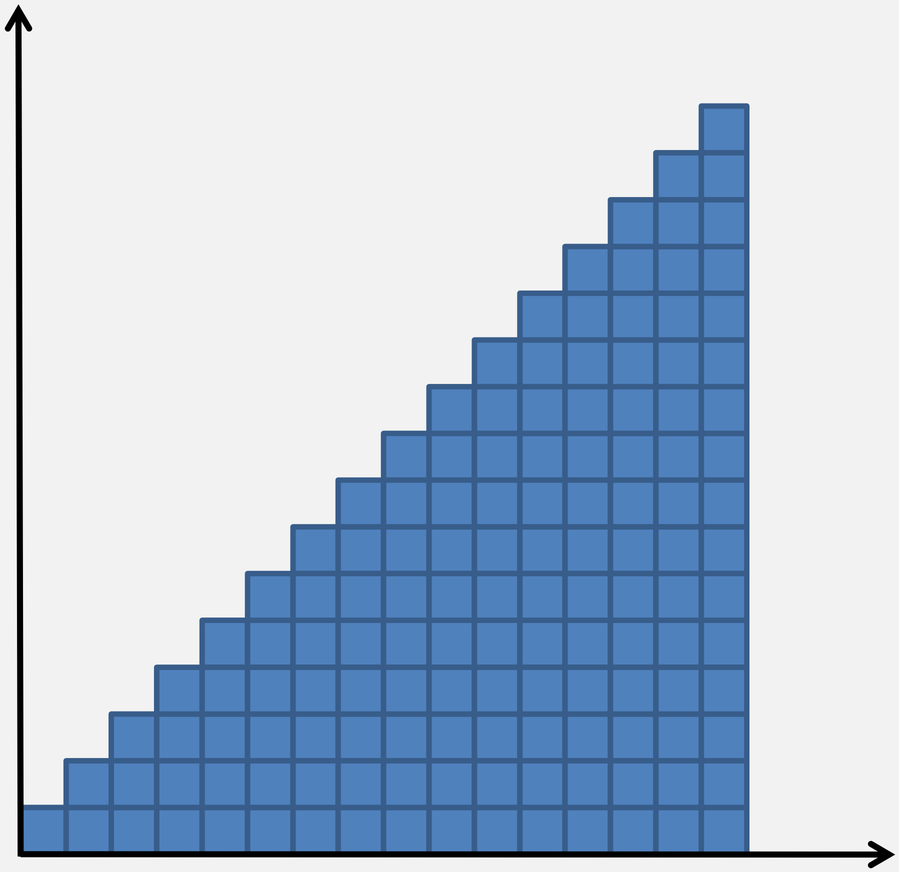
$$1+2+3+\ldots+n=\sum_{i=1}^{n} i$$



$$\sum_{i=1}^{n} i = \frac{(n+1)n}{2}$$

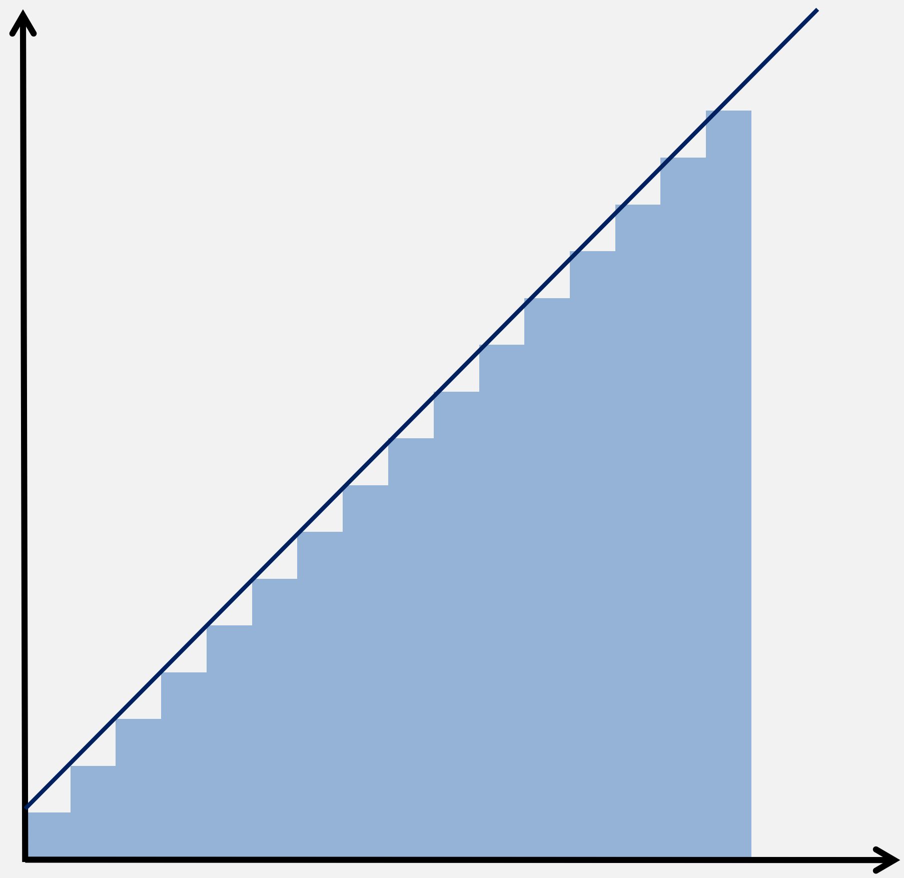
14

Bounding sums by integrals



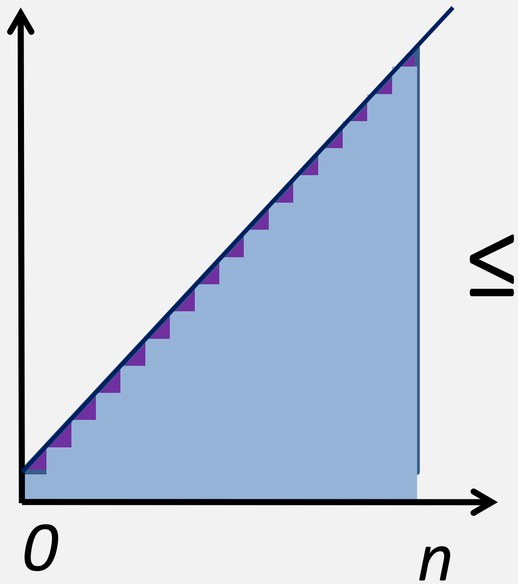
15

Bounding sums by integrals



16

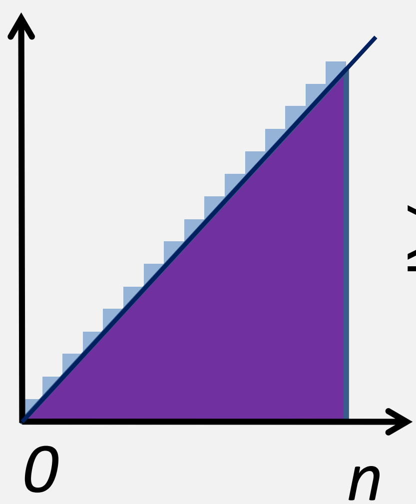
Bounding sums by integrals



$$\leq \int_0^n (x+1)\,dx = \left[\frac{(x+1)^2}{2}\right]_0^n$$

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

$\boxed{\sim n^2/2}$



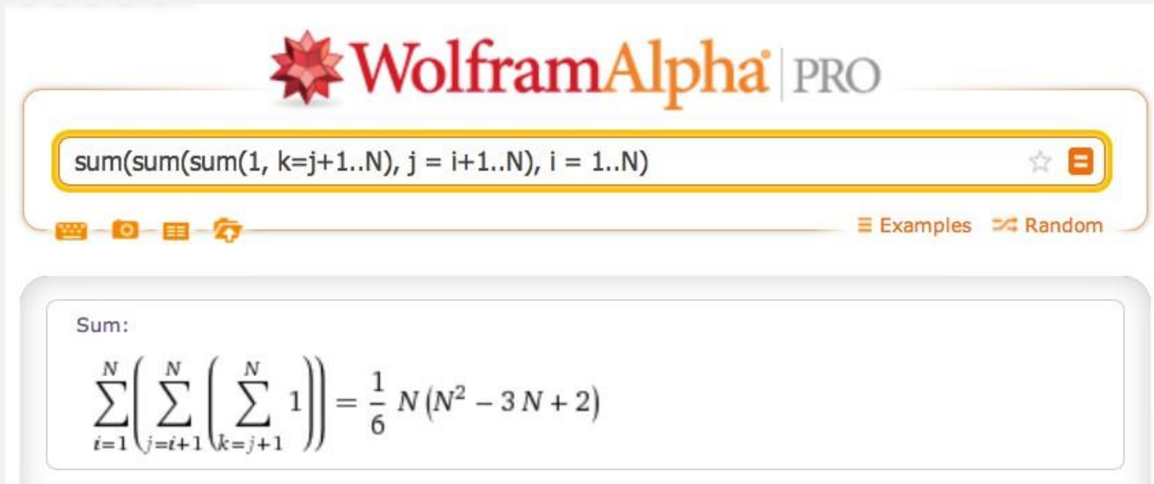
$$\geq \int_0^n x\,dx = \left[\frac{x^2}{2}\right]_0^n = n^2/2$$

17

Estimating a discrete sum

Q. How to estimate a discrete sum?

A3. Use Maple or Wolfram Alpha.



wolframalpha.com

```
[wayne:nobel.princeton.edu] > maple15
    |\^/|     Maple 15 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2011
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> factor(sum(sum(sum(1, k=j+1..n), j = i+1..n), i = 1..n));

                                n (n - 1) (n - 2)
                                -----------------
                                        6
```

Talning umferða

A.

```
sum = 0;
for (int i=0; i < n; i++)
  for (int j=0; j < i; j++)
    for (int k=0; k < j; k++)
       for (int l=0; l < k; l++)
           sum = sum + i*j*k*l;
```

Svar: $T(n) = \binom{n}{4} \sim n^4/24$

B.

```
sum = 0;
for (int i=0; i < n; i++)
 for (int j=0; j < n*n; j++)
     sum = sum + i*j;
```

Svar: $T(n) = n^3$

http://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

- *complexity of loops*
- *exponentials and logarithms*
- *order-of-growth classifications*
- dependencies on input
- memory
- summary of module 1

Exponentials and logarithms

Question. If you have 1 kr today, and double it every year, how long does it take to get to a million?

$$\underbrace{2 \cdot 2 \cdots 2}_{x \text{ times}} \geq 1.000.000$$

$$\Rightarrow \quad \boxed{\text{Find smallest } x \text{ such that } 2^x \geq 1.000.000}$$

Useful approximation. $10^3 = 1000 \cong 1024 = 2^{10}$

Easier version. If you have 1 kr today, and double it every year, how long does it take to get to a 1 mega kr?

$$\boxed{\text{Find smallest } x \text{ such that } 2^x \geq (1024)^2 = 2^{20}}$$

Exponentials and logarithms

Question. If you have 1 Giga kr today, and spend half of what you have each year, how long does it to become nothing?

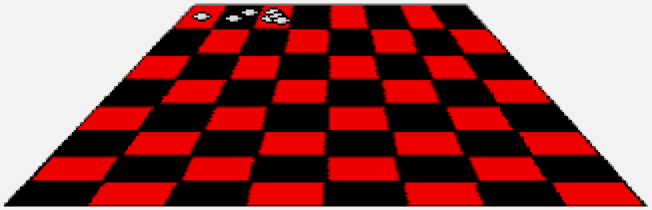
$$\frac{1\ G}{\underbrace{2 \cdot 2 \cdots 2}_{x \text{ times}}} \leq 1$$

$\Rightarrow$ $\boxed{\text{Find smallest } x \text{ such that } 2^x \geq 1G \sim 2^{30}}$

A classic fairytale

The ruler of India was so pleased with one of his wise men, who had invented the game of chess, that he offered this wise man a reward of his own choosing.

The wise man told his Master that he would like just one grain of rice on the first square of the chess board, double that number of grains of rice on the second square, and so on: double the number of grains of rice on each of the next 62 squares on the chess board.



How much rice is that? (1 grain of rice: $\frac{1}{64}g$, or $90mm^2$)

Exponentials and logarithms

$$\log(2^x) = x$$

$$2^{\log x} = x$$

How many times can we halve n until we get to 1?

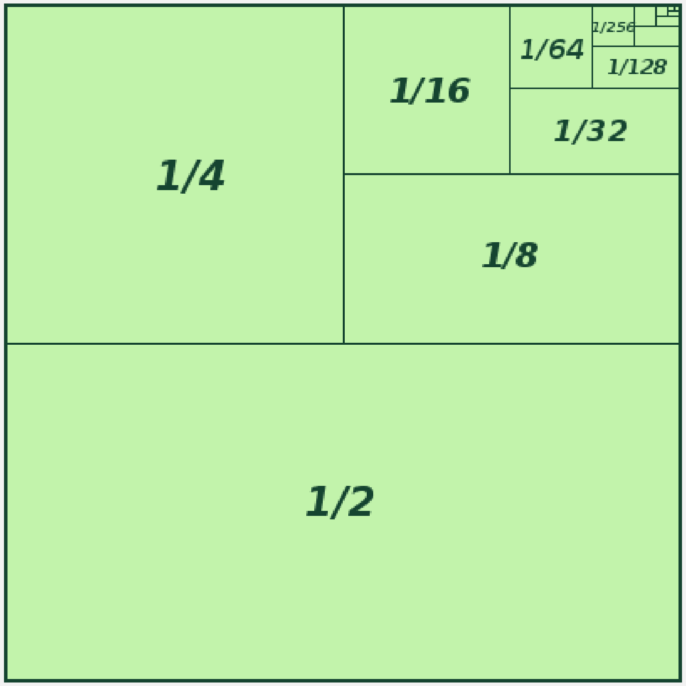
$2^k \rightarrow 2^{k-1} \rightarrow 2^{k-2} \rightarrow .... \rightarrow 2^2 \rightarrow 2^1 \rightarrow 2^0 = 1$
For $n = 2^k$ the answer is k.
Same for numbers in the range $2^k \ldots 2^{k+1}$ (if rounding down)

$\log(n)$ also counts how many bits are needed to represent the number n

Kvótaröð (Geometric series)

1 + 2 + 4 + 8 + 16 + ... + N = ??

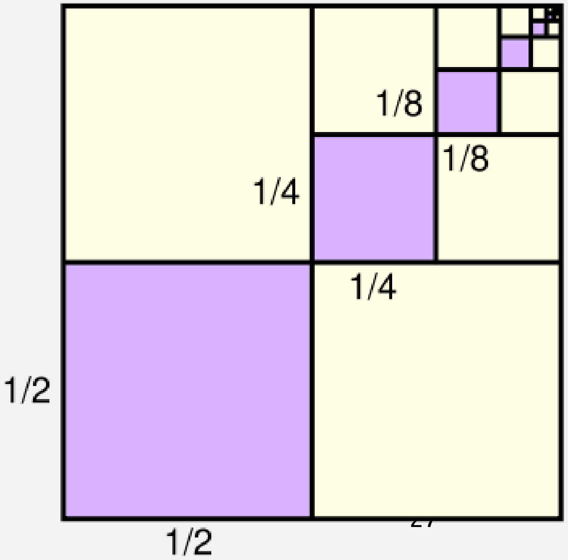
N + N/2 + N/4 + N/8 + ... + 1 = ??



$$\boxed{\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots=\sum_{n=0}^{\infty}\frac{1}{2}\left(\frac{1}{2}\right)^n=\frac{\frac{1}{2}}{1-\frac{1}{2}}=1.}$$

$$a+ar+ar^2+ar^3+ar^4+\cdots=\sum_{k=0}^{\infty}ar^k=\frac{a}{1-r}\Leftrightarrow|r|<1$$

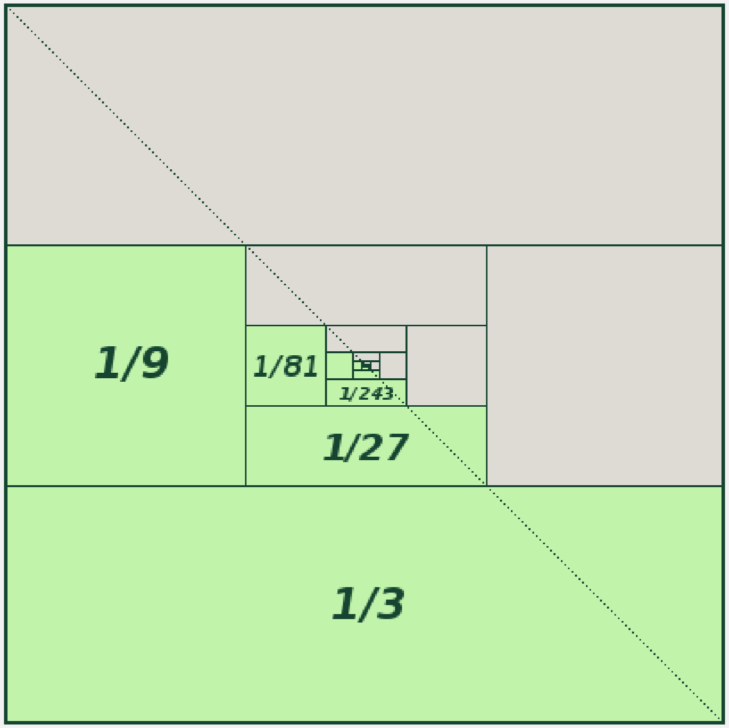
N/4 + N/16 + N/64 + ... + 1 = ??



More visual proofs

1 + 3 + 9 + … + N = ??

1/3 + 1/9 + … … = ??



Talning umferða

A.

```
sum = 0;
for (int i=0; i < n; i++)
 for (int j=1; j < n; j *= 2)
      sum = sum + i*j;
```

Svar: T(n) = n lg n

B.

```
sum = 0;
for (int i=0; i < n; i += i)
 for (int j=1; j < n; j *= 2)
      sum = sum + i*j;
```

Svar: $T(n) \sim (\lg n)^2$

C.

```
sum = 0;
for (int i=1; i < n; i += i)
 for (int j=1; j < n; j++)
      sum = sum + i*j;
```

Svar: (As stated): $T(n) \sim 2n$
(With i++): $T(n) \sim n \log n$

http://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

- *complexity of loops*
- *exponentials and logarithms*
- *order-of-growth classifications*
- dependencies on input
- memory
- summary of module 1

Common order-of-growth classifications

Definition. If $f(n) \sim c\, g(n)$ for some constant $c > 0$, then the order of growth of $f(n)$ is $g(n)$.

- Ignores leading coefficient.
- Ignores lower-order terms.

Ex. The order of growth of the running time of this code is n^3.

```
int count = 0;
for (int i = 0; i < n; i++)
   for (int j = i+1; j < n; j++)
      for (int k = j+1; k < n; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

Typical usage. Mathematical analysis of running times.

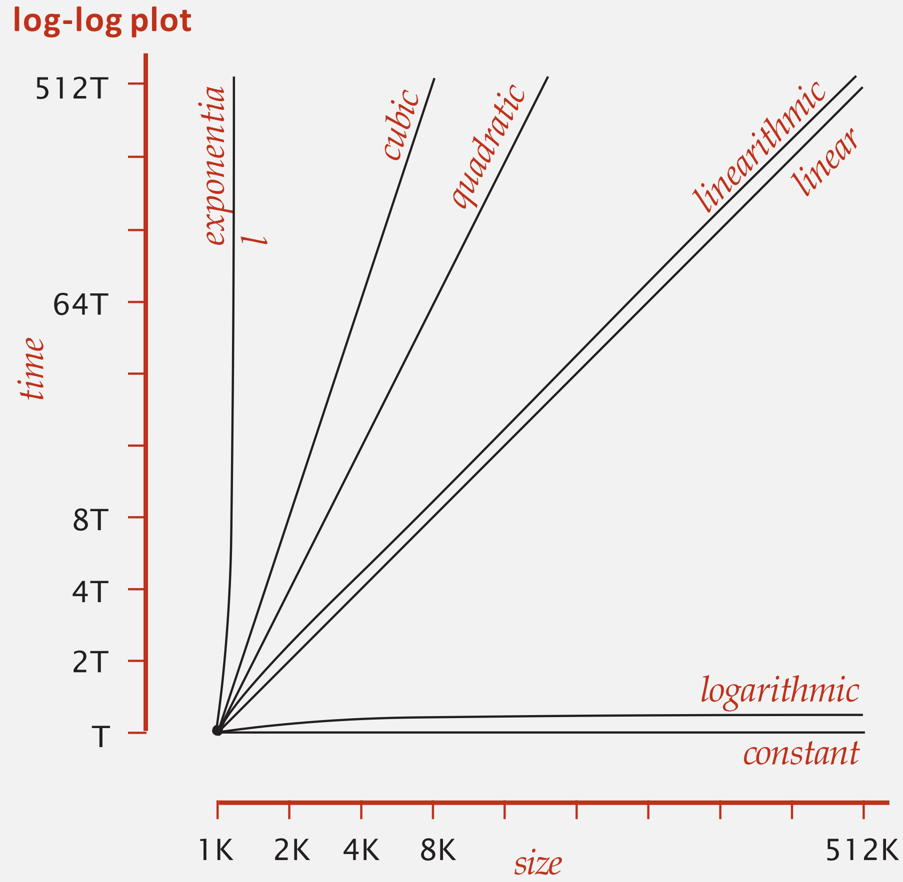
where leading coefficient depends on machine, compiler, JVM, ...

Common order-of-growth classifications

Good news. The set of functions

$$1, \ \log n, \ n, \ n \log n, \ n^2, \ n^3, \text{ and } 2^n$$

suffices to describe the order of growth of most common algorithms.



Typical orders of growth

Binary search

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

successful search for 33

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo							↑ mid							↑ hi

Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

successful search for 33

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo			↑ mid			↑ hi								

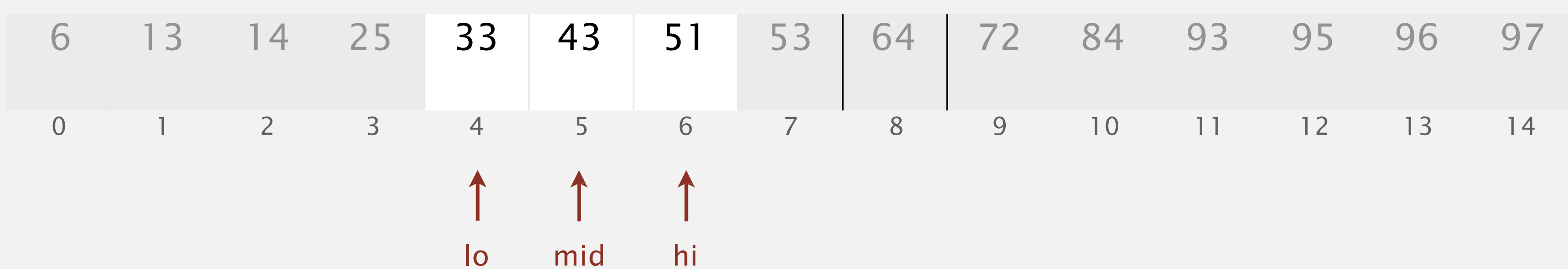
Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

successful search for 33



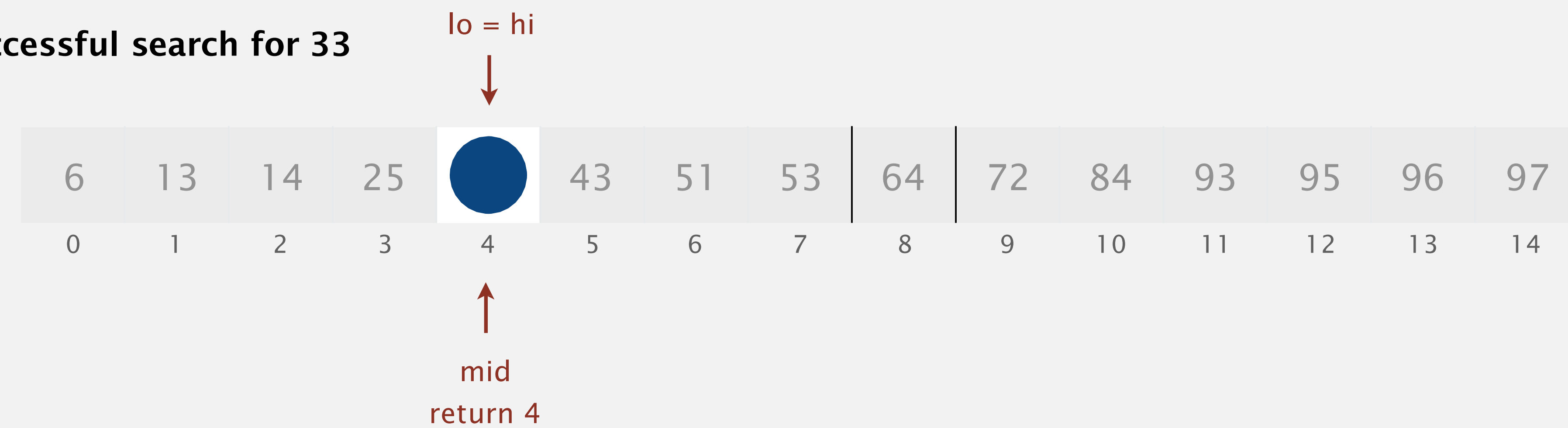
Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

successful search for 33



Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

unsuccessful search for 34

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo (0) mid (7) hi (14)

Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

unsuccessful search for 34

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo			↑ mid			↑ hi								

Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

unsuccessful search for 34

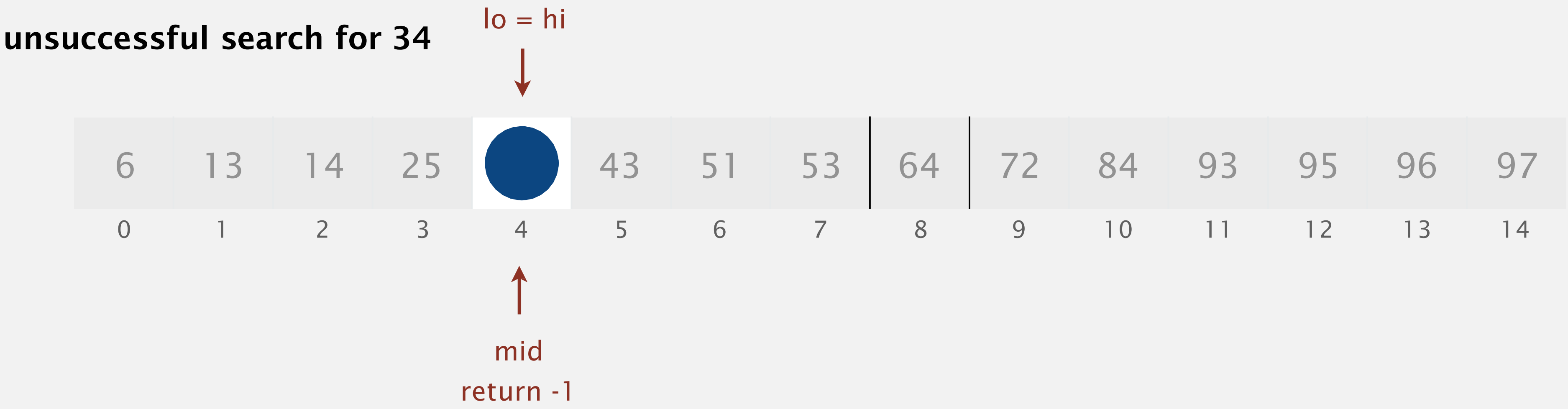
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
				lo	mid	hi								

Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.



Binary search: Java implementation

Invariant. If `key` appears in array `a[]`, then `a[lo]` ≤ `key` ≤ `a[hi]`.

Cost model. `key` comparisons. [Why?]

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length - 1;
   while (lo <= hi)
   {
      int mid = lo + (hi - lo) / 2;
      if      (key < a[mid]) hi = mid - 1;
      else if (key > a[mid]) lo = mid + 1;
      else return mid;
   }
   return -1;
}
```

why not mid = (lo + hi) / 2 ?

one "3-way compare"

Binary search : mathematical analysis

Proposition. Binary search uses at most $1 + \lg n$ key compares to search in a sorted array of size n.

Def. $T(n)$ = # key compares to binary search a sorted subarray of size $\le n$.

Binary search recurrence. $T(n) \le T(n/2) + 1$ for $n > 1$, with $T(1) = 1$.

left or right half (floored division)

possible to implement with one 2-way compare (instead of 3-way)

Pf sketch. [assume n is a power of 2]

$$
\begin{aligned}
T(n) &\le T(n/2) + 1 && \text{[given]} \\
&\le T(n/4) + 1 + 1 && \text{[apply recurrence to first term]} \\
&\le T(n/8) + 1 + 1 + 1 && \text{[apply recurrence to first term]} \\
&\;\vdots \\
&\le T(n/n) + \underbrace{1 + 1 + \ldots + 1}_{\lg n} && \text{[stop applying, } T(1) = 1 \text{]} \\
&= 1 + \lg n
\end{aligned}
$$

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

- *complexity of loops*
- *exponentials and logarithms*
- *order-of-growth classifications*
- dependencies on input
- memory
- summary of module 1

Types of analyses

Best case. Lower bound on cost.

- Determined by "easiest" input.
- Provides a goal for all inputs.

Worst case. Upper bound on cost.

- Determined by "most difficult" input.
- Provides a guarantee for all inputs.

Average case. Expected cost for random input.

- Need a model for "random" input.
- Provides a way to predict performance.

Ex 1. Array accesses for brute-force 3-SUM.

Best:	$\sim \frac{1}{2} N^3$
Average:	$\sim \frac{1}{2} N^3$
Worst:	$\sim \frac{1}{2} N^3$

Ex 2. Compares for binary search.

Best:	~ 1
Average:	$\sim \lg N$
Worst:	$\sim \lg N$

Types of analyses

Best case. Lower bound on cost.

Worst case. Upper bound on cost.

Average case. “Expected” cost.

Actual data might not match input model?

- Need to understand input to effectively process it.
- Approach 1: design for the worst case.
- Approach 2: randomize, depend on probabilistic guarantee.

Theory of algorithms

Goals.

- Establish “difficulty” of a problem.
- Develop “optimal” algorithms.

Approach.

- Suppress details in analysis: analyze “to within a constant factor”.
- Eliminate variability in input model by focusing on the worst case.

Optimal algorithm.

- Performance guarantee (to within a constant factor) for any input.
- No algorithm can provide a better performance guarantee.

Commonly-used notations in the theory of algorithms

notation	provides	example	shorthand for	used to
Tilde	leading term	$\sim 10n^2$	$10\,n^2$ $10\,n^2 + n\log n$ $10\,n^2 + 3n + 7$	provide approximate model
Big Theta	asymptotic growth rate	$\Theta(n^2)$	$0.3n^2$ $10\,n^2$ $22n^2 + n\log n + 3n$	classify algorithms
Big Oh	$\Theta(n^2)$ and smaller	$O(n^2)$	$10\,n^2$ $22\,n\log n + 3n$	develop upper bounds
Big Omega	$\Theta(n^2)$ and larger	$\Omega(n^2)$	$10\,n^2$ $10\,n^4$ $22n^3 + n\log n + 3n$	develop lower bounds

Common mistake. Interpreting big-Oh as an approximate model.

This course. Focus on approximate models: use Tilde-notation

Theory of algorithms: example 1

Goals.

- Establish "difficulty" of a problem and develop "optimal" algorithms.
- Ex. 1-SUM = "*Is there a* 0 *in the array?*"

Upper bound. A specific algorithm.

- Ex. Brute-force algorithm for 1-SUM: Look at every array entry.
- Running time of the optimal algorithm for 1-SUM is $O(N)$.

Lower bound. Proof that no algorithm can do better.

- Ex. Have to examine all N entries (any unexamined one might be 0).
- Running time of the optimal algorithm for 1-SUM is $\Omega(N)$.

Optimal algorithm.

- Lower bound equals upper bound (to within a constant factor).
- Ex. Brute-force algorithm for 1-SUM is optimal: its running time is $\Theta(N)$.

Theory of algorithms: example 2

Goals.

- Establish "difficulty" of a problem and develop "optimal" algorithms.
- Ex. 3-SUM.

Upper bound. A specific algorithm.

- Ex. Brute-force algorithm for 3-SUM.
- Running time of the optimal algorithm for 3-SUM is $O(N^3)$.

Theory of algorithms: example 2

Goals.

- Establish "difficulty" of a problem and develop "optimal" algorithms.
- Ex. 3-SUM.

Upper bound. A specific algorithm.

- Ex. Improved algorithm for 3-SUM.
- Running time of the optimal algorithm for 3-SUM is $O(N^2 \log N)$.

Lower bound. Proof that no algorithm can do better.

- Ex. Have to examine all N entries to solve 3-SUM.
- Running time of the optimal algorithm for solving 3-SUM is $\Omega(N)$.

Open problems.

- Optimal algorithm for 3-SUM?
- Subquadratic algorithm for 3-SUM?
- Quadratic lower bound for 3-SUM?

Algorithm design approach

Start.

- Develop an algorithm.
- Prove a lower bound.

Gap?

- Lower the upper bound (discover a new algorithm).
- Raise the lower bound (more difficult).

Caveats.

- Overly pessimistic to focus on worst case?
- Need better than "to within a constant factor" to predict performance.

Which of these statements are true?

A. $10n^3 = \Omega(n^2)$?

B. $10n^5 = O(n^3)$?

C. $10n^2 + n\log n = \Theta(n^2)$?

D. $n\log n = O(n^2)$?

E. $10n^2 = \Omega(n^2)$?

1.4 Analysis of Algorithms

- *complexity of loops*
- *exponentials and logarithms*
- *order-of-growth classifications*
- dependencies on input
- memory
- summary of module 1

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

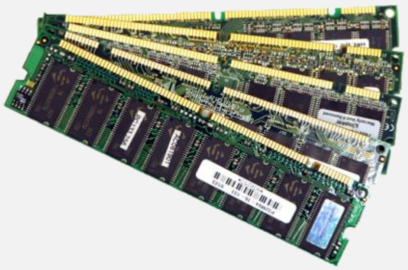
Basics

Bit. 0 or 1.

Byte. 8 bits.

Megabyte (MB). 2^{20} bytes (about 1 million).

Gigabyte (GB). 2^{30} bytes (about 1 billion).



64-bit machine. We assume a 64-bit machine with 8-byte pointers.

some JVMs "compress" ordinary object pointers to 4 bytes to avoid this cost

Typical memory usage for primitive types and arrays

type	bytes
`boolean`	1
`byte`	1
`char`	2
`int`	4
`float`	4
`long`	8
`double`	8

primitive types

type	bytes
`char[]`	$2n + 24$
`int[]`	$4n + 24$
`double[]`	$8n + 24$

one- dimensional arrays

type	bytes
`char[][]`	$\sim 2\,m\,n$
`int[][]`	$\sim 4\,m\,n$
`double[][]`	$\sim 8\,m\,n$

two- dimensional arrays

Typical memory usage for objects in Java

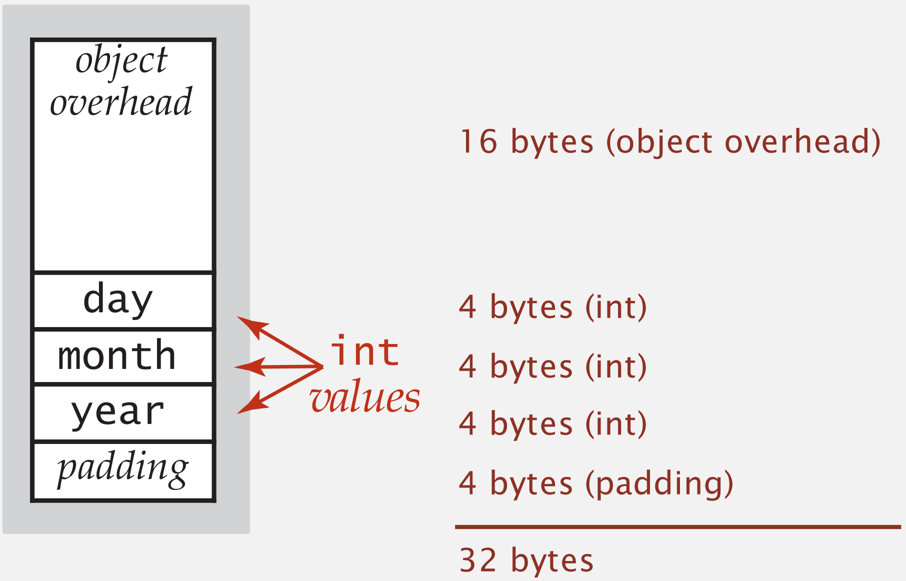
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Each object uses a multiple of 8 bytes.

Ex 1. A `Date` object uses 32 bytes of memory.

```
public class Date
{
   private int day;
   private int month;
   private int year;
...
}
```



Typical memory usage summary

Total memory usage for a data type value:

- Primitive type: 4 bytes for `int`, 8 bytes for `double`, …
- Object reference: 8 bytes.
- Array: 24 bytes + memory for each array entry.
- Object: 16 bytes + memory for each instance variable.
- Padding: round up to multiple of 8 bytes.

+ 8 extra bytes per inner class object
(for reference to enclosing class)

Note. Depending on application, we may want to count memory for any referenced objects (recursively).

Analysis of algorithms Quiz 3

How much memory does a `WeightedQuickUnionUF` use as a function of n ?

A. $\sim 4\,n$ *bytes*

B. $\sim 8\,n$ *bytes*

C. $\sim 4\,n^2$ *bytes*

D. $\sim 8\,n^2$ *bytes*

E. *I don't know.*

```
public class WeightedQuickUnionUF
{
   private int[] parent;
   private int[] size;
   private int count;

   public WeightedQuickUnionUF(int n)
   {
      parent = new int[n];
      size = new int[n];
      count = 0;
      for (int i = 0; i < n; i++)
          parent[i] = i;
      for (int i = 0; i < n; i++)
          size[i] = 1;
   }
   ...
}
```

Analysis of algorithms Quiz 3

How much memory does a `WeightedQuickUnionUF` use as a function of *n* ?

16 bytes
(object overhead)

8 + (4n + 24) bytes each
(reference + `int[]` array)

4 bytes (`int`)

4 bytes (padding)

8n + 88 ~ 8n bytes

```
public class WeightedQuickUnionUF
{
   private int[] parent;
   private int[] size;
   private int count;

   public WeightedQuickUnionUF(int n)
   {
      parent = new int[n];
      size = new int[n];
      count = 0;
      for (int i = 0; i < n; i++)
         parent[i] = i;
      for (int i = 0; i < n; i++)
         size[i] = 1;
   }
   ...
}
```

S1: Percolation

1. Implement the data type Percolation
   - Implement the API as specified
   - See Checklist for hints, further explanations
   - Good to test using the visualization clients supplied

2. Perform percolation experiments in the class PercolationStats.
   - Repeated random tests and statistics, as we have seen before

3. Write a report
   - Focus on answering the questions in the document X1.tex
   - Need not be very long; be to the point