Vector API:

```
public class Vector:
        Vector()       // býr til tóman vektor
void    add(Object o) // setur o aftast
int     size()           // skilar stærð
Object  get(int i) // stak í sæti i
```

Einföld notkun:

```
Vector v = new Vector();
while(!StdIn.isEmpty()) {
  v.add(new Integer(StdIn.readInt()));
}
…
int max = Integer.MIN_VALUE;
for (int i = 0; i < v.size(); i++) {
  Integer x = (Integer) v.get(i);
  max = Math.max(max, x.intValue());
}
```

Vektor útfærsla

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size

   public Vector() {
       this.size = 0;
       this.a = new Object[10];
   }

   public int size() {
       return this.size;
   }
}
```

Vektor útfærsla

Fastayrðing gagna, gríðarlega mikilvæg!

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size

   public Vector() {
       this.size = 0;
       this.a = new Object[10];
   }

   public int size() {
       return this.size;
   }
}
```

Vektor útfærsla

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size
   ... // framhald
   public void add(Object o) {
       if (size >= a.length) {
          Object[] tmp = new Object[2*a.length];
          … // afrita frá a yfir í tmp
          a = tmp;
       }
       a[size] = o;
       size++;
   }
}
```

Vektor útfærsla

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size
   ... // framhald
   public Object get(int i) {
       if (i < 0 || i >= size) {
          throw new ArrayIndexOutOfBoundsException(i);
       }
       return a[i];
   }
}
```

Vektor útfærsla

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size
   ... // framhald
   public Object get(int i) {
       if (i < 0 || i >= size) {
          throw new ArrayIndexOutOfBoundsException(i);
       }
       return a[i];
   }
}
```

Krassar forritinu með
“hjálplegum” villuskilaboðum

Vektorar

```
public class Vector:
        Vector()       // býr til tóman vektor
void    add(Object o) // setur o aftast
int     size()           // skilar stærð
Object  get(int i) // stak í sæti i
```

Gamla Java (1.1-1.4) þurfum alltaf að kasta Object yfir í eitthvað annað. Í 99% tilvika eru allir hlutirnir í vektornum af sama tagi (eða yfirklasa).

Viljum geta búið til Vector af Integer, Vector af String, Vector af hverju sem er

Sniðmát

Sniðmát (generics) leysa þennan vanda. Skilgreining á klasa getur tekið inn aðra klasa sem “viðföng” með

```
public class A<E> {…}
```

Þá er A klasi sem að veit ekki fyrirfram hvað E er en má skilgreina tilvísanir á hluti af taginu E. Til að búa til tilvik af klasanum A þarf að taka fram hvað E er, t.d.

```
A<String> a = new A<String>();
```

`A<String>` er þá tagið á nýja hlutnum og kóðinn virkar rétt eins og við hefðum skipt E út fyrir String

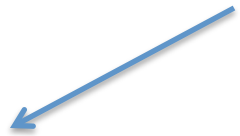
Sniðmát

```
public class Vector<E>:
        Vector()      // býr til tóman vektor
void    add(E e)// setur e aftast
int     size()           // skilar stærð
E       get(int i) // stak í sæti i
```

Klasinn E kemur fyrir í add() þar sem við getum bara bætt við E hlutum og í get() þar sem vektorinn skilar bara E hlutum.

Sama dæmi og áðan

Java 1.7+ fattar að þetta er Vector<Integer>



```
Vector<Integer> v = new Vector<>();
while(!StdIn.isEmpty()) {
  v.add(new Integer(StdIn.readInt()));
}
…
int max = Integer.MIN_VALUE;
for (int i = 0; i < v.size(); i++) {
  max = Math.max(max, v.get(i).intValue());
}
```

Sama dæmi og áðan

```
Vector<Integer> v = new Vector<>();
while(!StdIn.isEmpty()) {
  v.add(new Integer(StdIn.readInt()));
}
…
int max = Integer.MIN_VALUE;
for (int i = 0; i < v.size(); i++) {
  max = Math.max(max, v.get(i).intValue());
}
```

v.get(i) skilar Integer, ekkert óþarfa kast

Greining forrita

Góð forrit eiga að keyra rétt, hratt og með sem minnstum “auðlindum”.

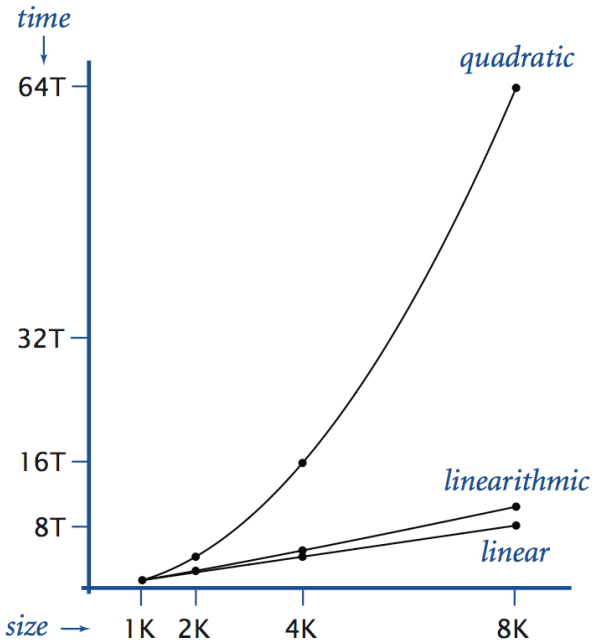
- Er hægt að leysa verkefnið á hraðvirkari hátt?
- Hversu hratt er forritið mitt?
- Hversu mikil áhrif hefur þessi breyting á forritið?
- Hve mikið minni/disk/net notar forritið mitt?

Greining á forritun leyfir okkur að svara þessum spurningum án þess að forrita/prófa/mæla

Greining forrita

Röðun

- Endurraða N stökum í vaxandi röð
- Notkun: Gagnagrunnar, tölfræði, lífupplýsingafræði, biðraðastjórnun
- Einföld lausn: N^2 skref (insertion sort, bubble sort)
- Betra reiknirit: N log(N) skref (merge sort, quick sort)



amazon.com

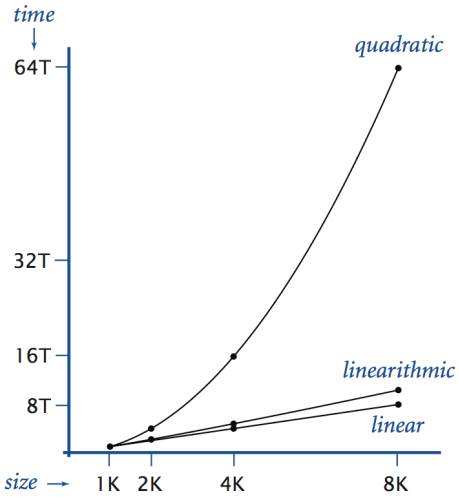
Google

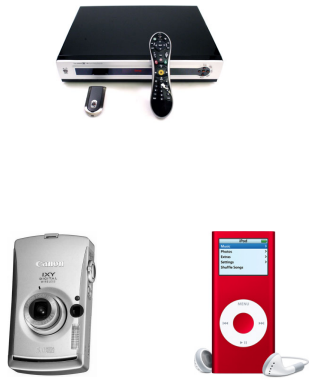
ebay

Greining forrita

Strjál Fourier ummyndun

- Brýtur bylgju niður í reglulega þætti
- Hagnýting: DVD, JPEG, MP3, margföldun
- Einföld lausn: N^2 skref
- Betri lausn: N log(N) skref, FFT (Fast Fourier Transform)





Þrjú atriði sem eru orðin mikilvæg í dag hjá hugbúnaðarfyrirtækjum

1. Reiknirit (Algorithms)
2. Reiknigreind (Machine learning)
3. Dreifð kerfi (Distributed Systems)

Google, Amazon, eBay, Netflix, Facebook og fleiri byggja kjarnann sinn á þessum þremur sviðum

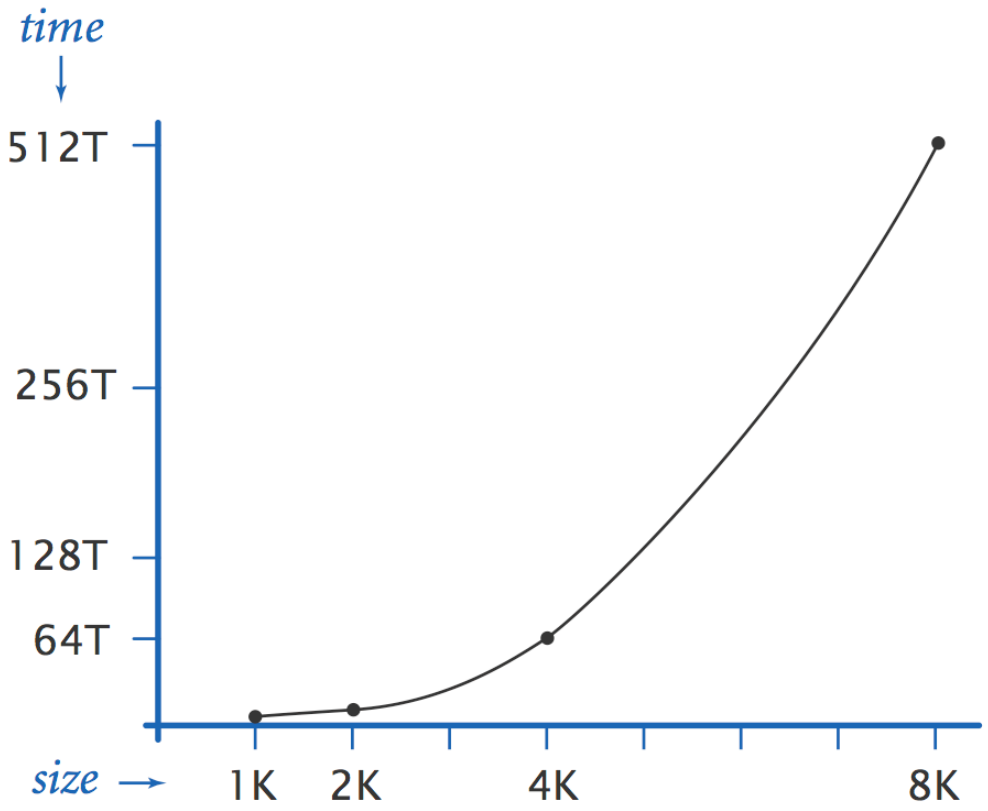
Greining forrita

Hvernig greinum við forrit? T.d. hraða

- Mælum tímann sem það tekur?
- Nokkrum sinnum?
- Háð inntakinu
- Oftast háð stærðinni á inntakinu
  - Mælum tímann fyrir mism. stór inntök

N	*ttími (sek)*
512	0.03
1,024	0.26
2,048	2.16
4,096	17.18
8,192	136.76

Plottum gögnin



Hversu hratt eykst keyrslutíminn sem fall af stærð inntaks N?

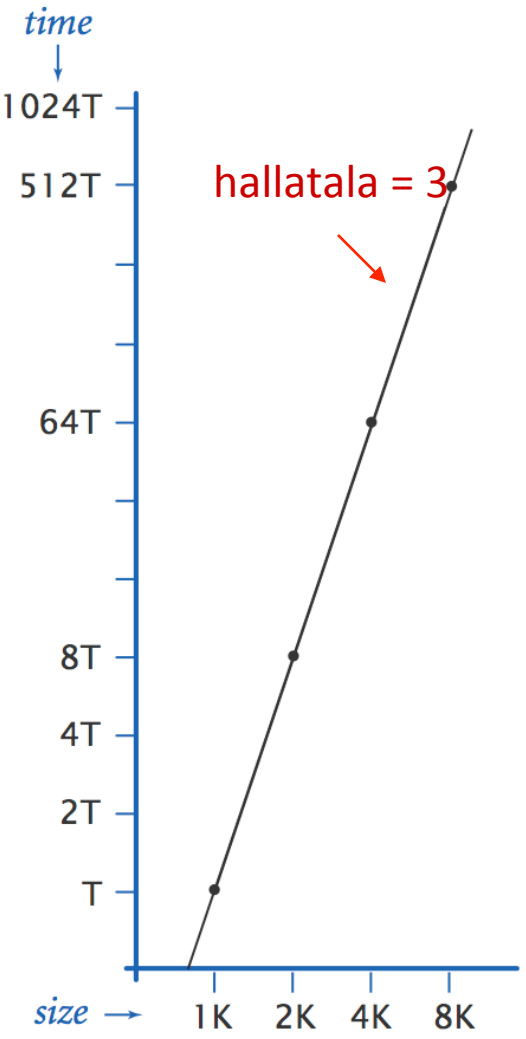
Greining forrita

Giskum á keyrslu tímann

$T(N) = aN^b$

Greinum gögning: plottum tíma vs. inntaksstærð N á log-log skala

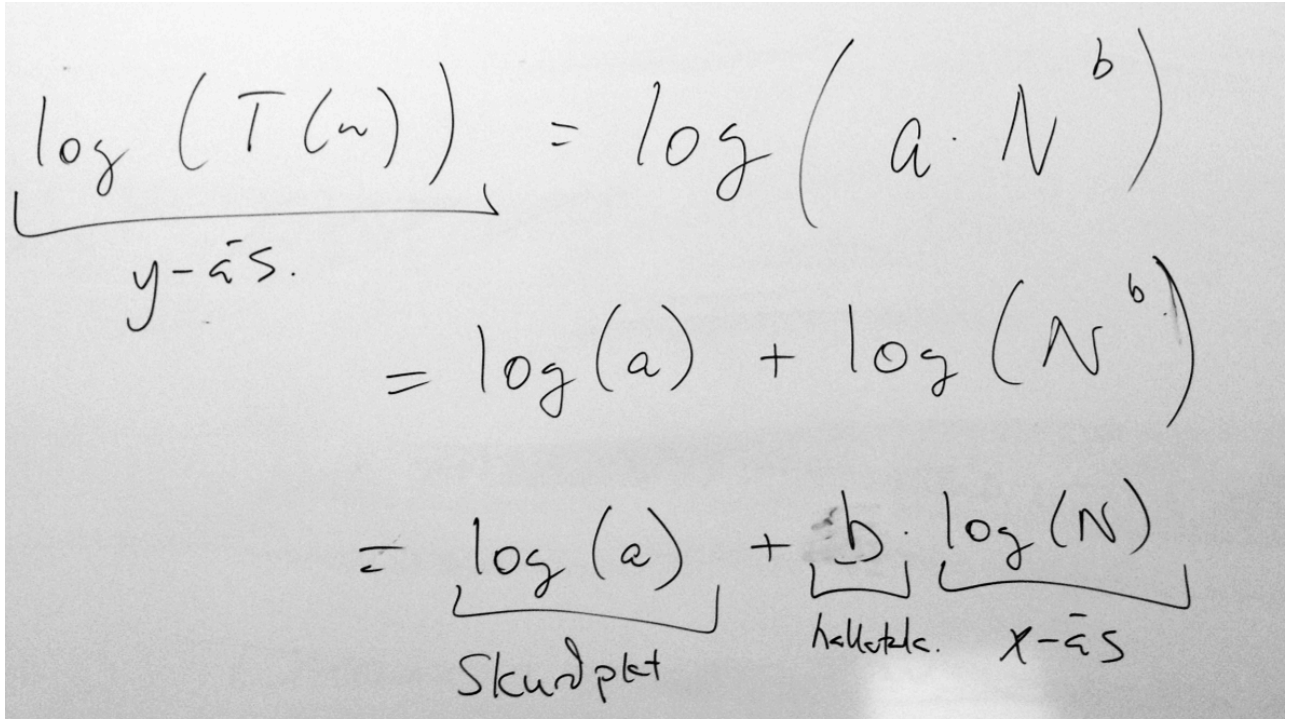
Margliða leiðir til beinnar línu, með hallatölun b



Log-log plot

Giskum á keyrslu tímann

$T(N) = aN^b$



Finnum b sem hallatölu grafsins.

Greining forrita

Þessi leið er frekar takmörkuð til að spá fyrir um hraða

Viljum líka geta talað um hraða forrits óháð tölvunni sem keyrir það

Teljum fjölda frum-aðgerða

- skilgreining breyta
- gildisveiting
- samanburður
- aðgerðir á frumstæðum tögum
- vísanir í fylki

Leggjum saman fjölda aðgerða

```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0) count++;
```

operation	frequency
Skilgreining	2
gildisveiting	2
< samanburður	$N + 1$
== samanburður	N
vísun í fylki	N
hækkun	$\leq 2N$

fer eftir því hve oft a[i]==0, í mesta lagi 2N, a.m.k N

Samtals í mesta lagi 5N+5 aðgerðir.

Greining forrita

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

operation	frequency
skilgreiningar	$N + 2$
gildisveitingar	$N + 2$
< samanburður	$1/2\ (N + 1)\ (N + 2)$
== samanburður	$1/2\ N\ (N - 1)$
fylki	$N\ (N - 1)$
hækkun	$\leq\ N^2$

$$(N-1) + \ldots + 2 + 1 + 0 = 1/2\ N(N-1)$$

mjög seinlegt að telja nákvæmlega

“Tilde” notation

Slump

- Metum keyrslutíma sem fall af stærð inntaks N
- sleppum minni liðum
  - þegar N er stórt skipta litlu liðirnir engu máli
  - þegar N er lítið skiptir heildin engu máli

Dæmi1. $6N^3 + 17N^2 + 56 \sim 6N^3$

Dæmi2. $6N^3 + 100N^{4/3} + 56 \sim 6N^3$

Dæmi3. $6N^3 + 17N^2 \log N \sim 6N^3$

hendum minni liðum

(t.d., N = 1000: 6 miljarðar vs. 169 miljónir)

Skilgreining $f(N) \sim g(N)$ þýðir $\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1$

O - notation

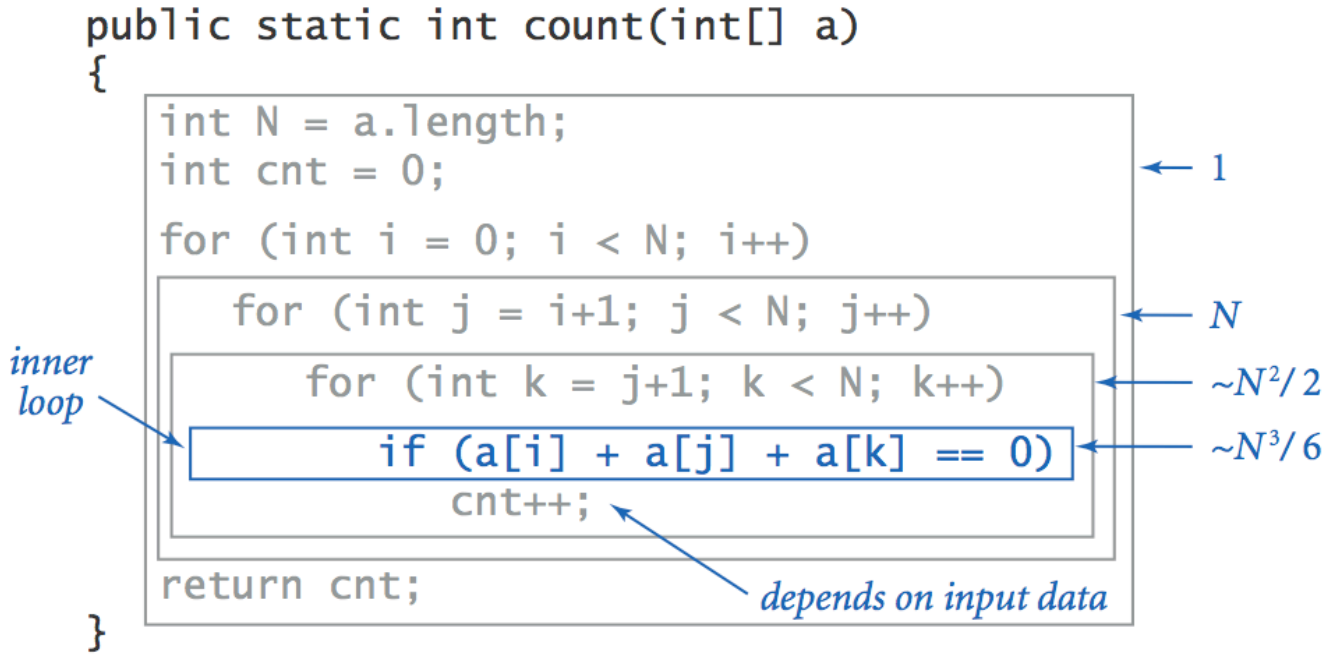
~ er ágætt ef við viljum nákvæm svör

- Við getum ekki búist við nákvæmum svörum
  - Teljum fjölda aðgerða, aðgerðir eru misdýrar
  - Mismunandi eftir örgjörvum
  - Háð því hvað þýðandinn gerir
  - Hvað er að gerast á tölvunni
- Segir ekki nákvæmlega til um keyrslutíma

O( ) – notation, sleppir stuðlinum við hæsta lið

- $6N^3$ verður $O(N^3)$
- $N^2 + N \log(N)$ verður $O(N^2)$
- 14 verður O(1) – fasti óháður N

Dæmi



Teljum fjölda aðgerða

- Fyrsta for lykkja keyrð O(N) sinnum
- Næsta for lykkja keyrð O(N*N) sinnum
- Þriðja for lykkjan keyrð O(N*N*N) sinnum
- Innra if O(1) í hvert skipti, samtals O(N*N*N)
- Samtals $O(N^3)$

Föll og aðferðir

Teljum 1 (eða O(1)) aðgerð fyrir frumstæðar aðgerðir.

Þegar við köllum á föll, aðferðir eða smið teljum við fjölda aðgerða í því falli

Dæmi:

```
int[] a = new int[N];
```
// kostar O(N) aðgerðir

```
int max = find_max(a);
```
// skoðar allt fylkið O(N)

```
Arrays.sort(a);
```
// O(N log(N)) skv. lýsingu

Æfingar

1.

```
while (N > 1) {
   N = N / 2;
   ...
}
```

svar: $O(\log_2(N))$

2.

```
for (int i = 0; i < N; i++)
   ...
```

svar: $O(N)$

3.

```
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      ...
```

svar: $O(N^2)$

Æfingar

4.

```
public static void g(int N) {
   if (N == 0) return;
   g(N/2);
   g(N/2);
   for (int i = 0; i < N; i++)
      ...
}
```

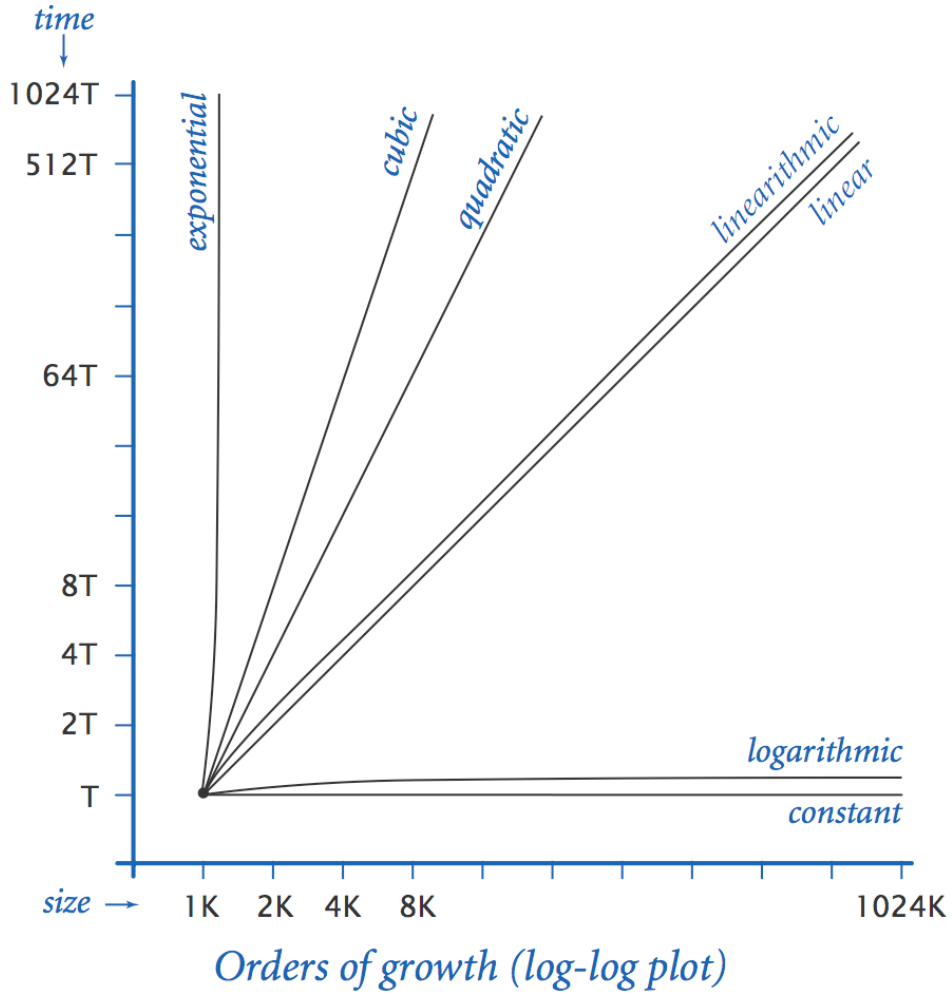
svar: O(Nlog(N))

5.

```
public static void f(int N) {
   if (N == 0) return;
   f(N-1);
   f(N-1);
   ...
}
```

svar: $O(2^N)$

Túlkun á O(…)



Orders of growth (log-log plot)

order of growth *description*	*function*	*factor for doubling hypothesis*
constant	1	1
logarithmic	$\log N$	1
linear	N	2
linearithmic	$N \log N$	2
quadratic	N^2	4
cubic	N^3	8
exponential	2^N	2^N

Commonly encountered growth functions

Gróf röðun: $O(1) << O(\log(N)) << O(N^a) << O(a^N)$

Túlkun á O(…)

order of growth	*predicted running time if problem size is increased by a factor of 100*
linear	a few minutes
linearithmic	a few minutes
quadratic	several hours
cubic	a few weeks
exponential	forever

Effect of increasing problem size for a program that runs for a few seconds

order of growth	*predicted factor of problem size increase if computer speed is increased by a factor of 10*
linear	10
linearithmic	10
quadratic	3-4
cubic	2-3
exponential	1

Effect of increasing computer speed on problem size that can be solved in a fixed amount of time