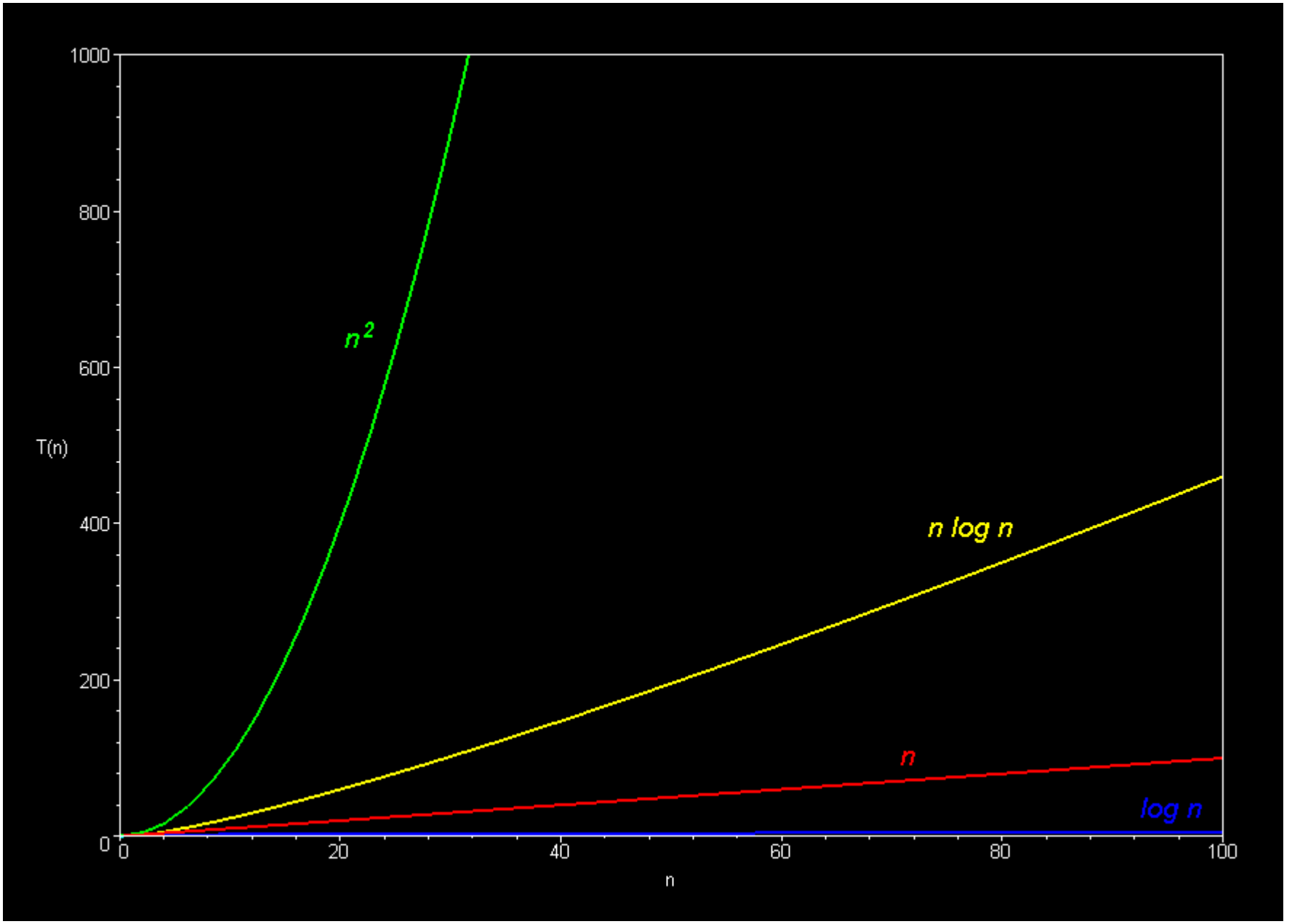
En anvendelse av køer:
Radixsortering

Hvor raskt er det mulig å sortere?

- Sortering av *n* tall kan aldri bli raskere enn $O(n)$:
  - Bare det å sjekke at tallene er sortert krever en full gjennomgang av alle *n* tall
- Kan bevises at:
  - Sortering av *n* elementer basert på sammenligning av to og to verdier aldri kan gjøres raskere enn $O(n \log n)$
- Det finnes allikevel sorteringsalgoritmer som er $O(n)$:
  - Bruker andre kriterier enn sammenligning av to og to verdier for å flytte rundt på tallene i sorteringen
  - Algoritmene krever at vi på forhånd har noe mer informasjon om tallene som skal sorteres


1000
800
600
400
200
0
T(n)
0
20
40
60
80
100
n
n^2
n log n
n
log n

En enkel *O*(*n*) metode: Counting sort

- Skal sortere en array *A* med *n* ikke-negative heltall
- Vet at alle tallene i *A* er mindre enn et tall *m*
- Algoritme:
  - Opprett en array *B* av lengde *m* fylt med nuller
  - Gå gjennom *A* én gang, tell opp antall ganger hver av verdiene 0, 1, 2, 3, …, *m* - 1 forekommer, lagre antall ganger en verdi *x* forekommer i *B*(*x*)
  - Gå gjennom B én gang og fyll opp *A* med *B*(0) 0'er, *B*(1) 1'ere, *B*(2) 2'ere, …, *B*(*m* - 1*)* verdier lik *m* - 1

Counting sort: Eksempel

$n = 14$
$m = 5$

A input

0	4	2	2	0	0	1	1	0	1	0	2	4	2

B

0	1	2	3	4
5	3	4	0	2

A output

0	0	0	0	0	1	1	1	2	2	2	2	4	4

Counting sort: Effektivitet

- Opptelling av antall forekomster av verdier i A : $O(n)$
- Gjennomgang av B: $O(m)$
- Totalt: $O(m + n)$
- Counting sort er $O(n)$ hvis m ikke er veldig mye større enn n
- Implementasjon av counting sort: Øvingsoppgave

Counting sort: Anvendbarhet

- Enkle metoder som counting sort er et godt alternativ for sortering av mange heltall med relativt liten variasjon i verdiene
- Problemer:
  - Krever $O(m)$ ekstra minne i tillegg til arrayen A som skal sorteres – ubrukbar til f.eks. generell sortering av 32-bits heltall, $m = 2^{32} = 4\ 294\ 967\ 296$
  - Kan bare sortere heltall, bokstaver og andre diskrete data, kan ikke brukes for f.eks. floating-point tall

Radix*-sortering

- For å sortere en array *A* med *n* ikke-negative heltall
- Tallene som skal sorteres har maksimalt *m* siffer
- Samme prinsipp som i maskiner for sortering av hullkort
- Går gjennom de *n* tallene *m* ganger, hver gang sorteres tallene på ett bestemt siffer
- Begynner med å sortere på siste siffer (enerne), deretter på nest siste siffer (tierne), osv.
- I hver gjennomløp fordeles tallene på 10 køer, en for hvert av sifrene 0, 1, 2, 3, …, 9, for å sikre at de hele tiden ligger i riktig rekkefølge

* Radix: Antall siffer i et tallsystem, f.eks. 10 for desimale tall og 2 for binære

Radixsortering: Eksempel

$n = 11$, $m = 3$

A = (324, 8, 216, 512, 27, 729, 0, 1, 343, 125, 64)

Første gjennomløp, sorterer på siste siffer i hvert tall:

Kø	0	1	2	3	4	5	6	7	8	9
Tall	0	1	512	343	324 64	125	216	27	8	729

Legger køene tilbake i A i sortert rekkefølge:

A = (0, 1, 512, 343, 324, 64, 125, 216, 27, 8, 729)

Radixsortering: Eksempel forts.

A = (0, 1, 512, 343, 324, 64, 125, 216, 27, 8, 729)

Andre gjennomløp, sorterer på nest siste siffer i hvert tall, for tall med mindre enn tre siffer brukes 0: 1 = 001, 27 = 027 ...

Kø	0	1	2	3	4	5	6	7	8	9
Tall	0 1 8	512 216	324 125 27 729		343		64			

Legger køene tilbake i A i sortert rekkefølge:

A = (0, 1, 8, 512, 216, 324, 125, 27, 729, 343, 64)

Radixsortering: Eksempel forts.

A = (0, 1, 8, 512, 216, 324, 125, 27, 729, 343, 64)

Tredje gjennomløp, sorterer på første siffer i hvert tall:

Kø	0	1	2	3	4	5	6	7	8	9
Tall	0 1 8 27 64	125	216	324 343		512		729		

Legger køene tilbake i A i sortert rekkefølge, sortering ferdig:

A = (0, 1, 8, 27, 64, 125, 216, 324, 343, 512, 729)

Radix-sortering av tegnstrenger

rat	mop	map	car
mop	map	rap	cat
cat	top	car	cot
map	rap	tar	map
car	car	rat	mop
top	tar	cat	rap
cot	rat	mop	rat
tar	cat	top	tar
cap	cot	cot	top

- Krever like mange køer som antall mulige tegn/bokstaver
- Implementasjon: Øvingsoppgave

Radixsortering av heltall: Algoritme

Input: *A*: Array med *n* ikke-negative heltall
m: Maks. antall siffer i tallene i A

1 Opprett en array *Q* med 10 tomme køer

2 For $k = 0, 1, 2, \ldots, m - 1$

2.1 For $i = 0, 1, 2, \ldots, n - 1$

2.1.1 $index = (A[i] / 10^{k}) \% 10$

2.1.2 *Q[index].enqueue(A[i])*

2.2 $i = 0$

2.3 For $j = 0, 1, 2, \ldots, 9$

2.3.1 While *!Q[j].isEmpty( )*

2.3.1.1 *A[i] = Q[j].dequeue( )*

2.3.1.2 $i = i + 1$

Radixsortering: Effektivitet

- *m* gjennomløp av de *n* tallene som skal sorteres
- Hvert gjennomløp har lineær arbeidsmengde – innsetting av *n* tall i kø og sammenslåing av køene til slutt
- Total arbeidsmengde: $O(m \cdot n)$
- Vanligvis er *n* mye større enn *m*, arbeidsmengde er da $O(n)$
- Brukes ikke mye i praksis, blir treg fordi algoritmen hele tiden må håndtere innsetting i og sammenslåing av køer
- Fungerer bare for heltall og andre typer diskrete data der vi kjenner maks antall "siffer"
- Implementasjon i Java