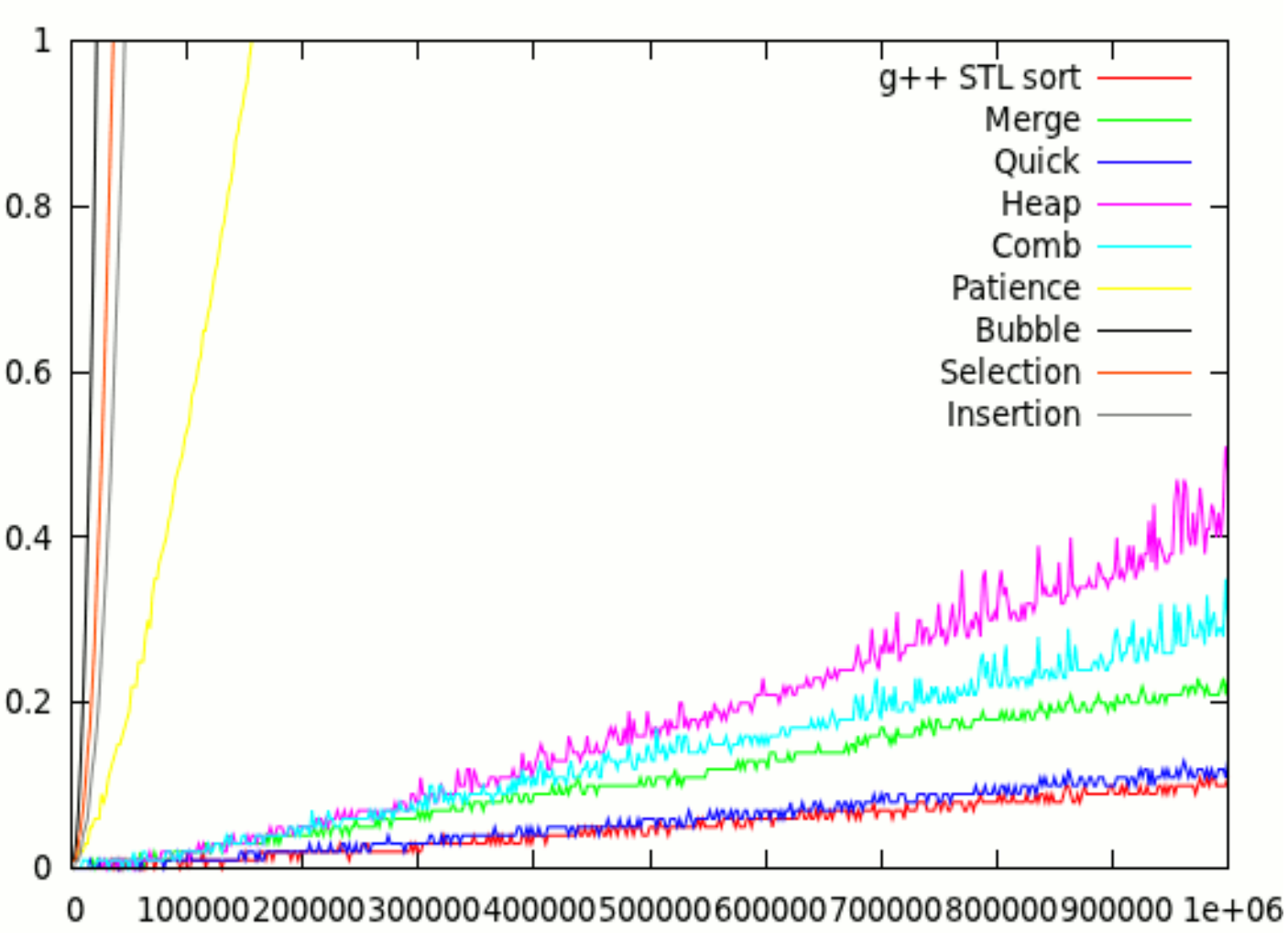
# Heapsort


1
0.8
0.6
0.4
0.2
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1e+06
g++ STL sort
Merge
Quick
Heap
Comb
Patience
Bubble
Selection
Insertion

# Heapsort

Algoritme for å sortere array med *n* elementer:

1. Sett alle elementene inn i en heap som initielt er tom
2. Ta ut minste element av heap'en *n* ganger og sett dem tilbake i arrayen i sortert rekkefølge

- Enkel heapsort med heltall: `heapSortDemo.java`

# Heapsort: Effektivitet

- Arbeidsmengde:
  - Gjør først $n$ innsettinger i en heap som initielt er tom: $O(n \log(n))$ fordi heapen alltid er balansert
  - Deretter $n$ fjerninger av minste element i heap, også $O(n \log(n))$
  - Totalt: $O(n \log(n))$
- Fordel:
  - Garantert $O(n \log(n))$, ikke avhengig av data
- Ulempe:
  - Bruker $O(n)$ ekstra memory til heap

# In-place heapsort *

- Bruker *ikke* en ekstra array til å lagre heap
- Både innsetting og fjerning i heap gjøres *inne i* opprinnelig array (in-place)
- Bruker en *max-heap*:
  - Forelder er alltid *større eller lik* barna
  - *Største* verdi ligger i roten
- Strategi:
  - Gjør om hele arrayen til en max-heap
  - Fjern første verdi fra heap og legg den på indeksen rett etter heap, inntil heap er tom

*: J.W.J. Williams: "Algorithm 232: Heapsort", *Communications of the ACM*, 1964

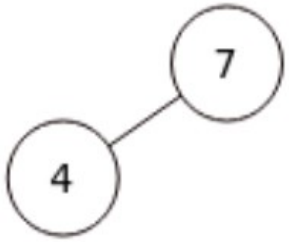
# Bruker to hjelpefunksjoner

- `siftUp(int A[], int i)`
  - `A[0]`, `A[1]`,... , `A[i-1]` er en max-heap
  - Setter verdien `A[i]` inn i heap ved å bytte den oppover med foreldernode inntil den står riktig
- `siftDown(int A[], int i)`
  - `A[0]`, `A[1]`,... , `A[i]` er en max-heap, muligens med unntak av roten `A[0]` som kan stå feil
  - Setter verdien `A[0]`på riktig plass i heap ved å bytte den nedover med største barn inntil den står riktig

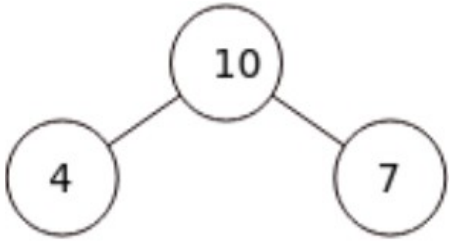
# Eksempel : Del 1 – Bygger max-heap

4 7 10 9 5

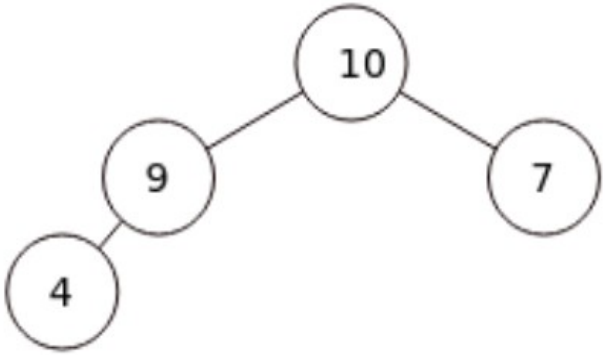

siftUp 7 4 7 10 9 5



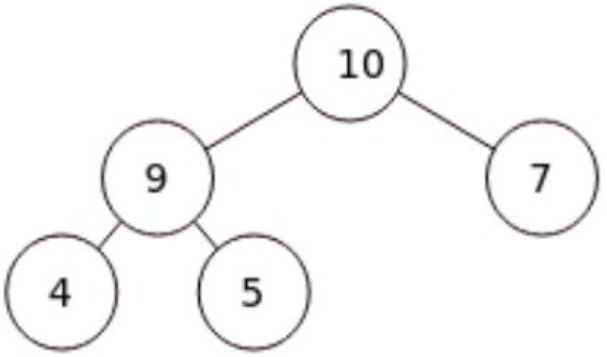

siftUp 10 7 4 10 9 5




siftUp 9 10 4 7 9 5



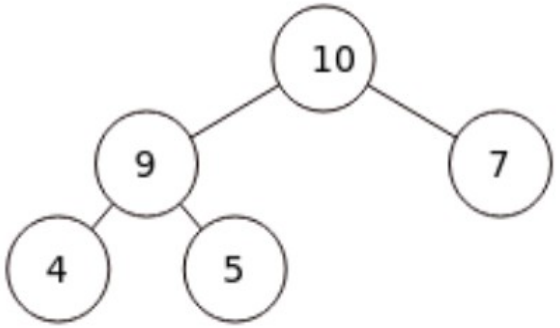

siftUp 5 10 9 7 4 5




10 9 7 4 5

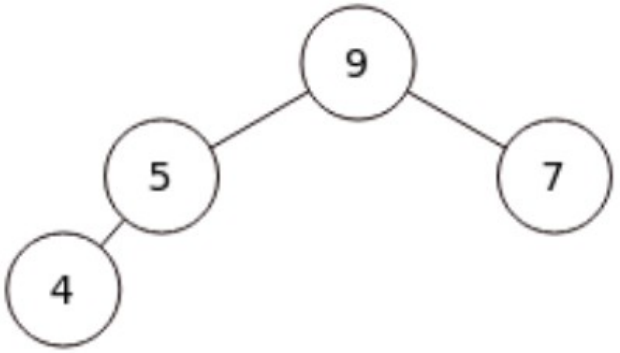
# Eksempel : Del 2 – Ta ut største verdi og sett bakerst

10 9 7 4 5



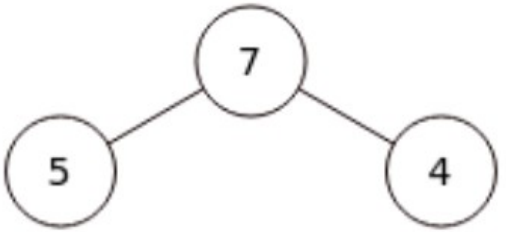

swap 10 og 5 5 9 7 4 10

siftDown 5 9 5 7 4 10



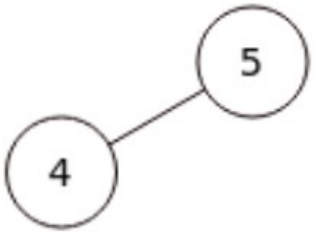

swap 9 og 4 4 5 7 9 10

siftDown 4 7 5 4 9 10




swap 7 og 4 4 5 7 9 10

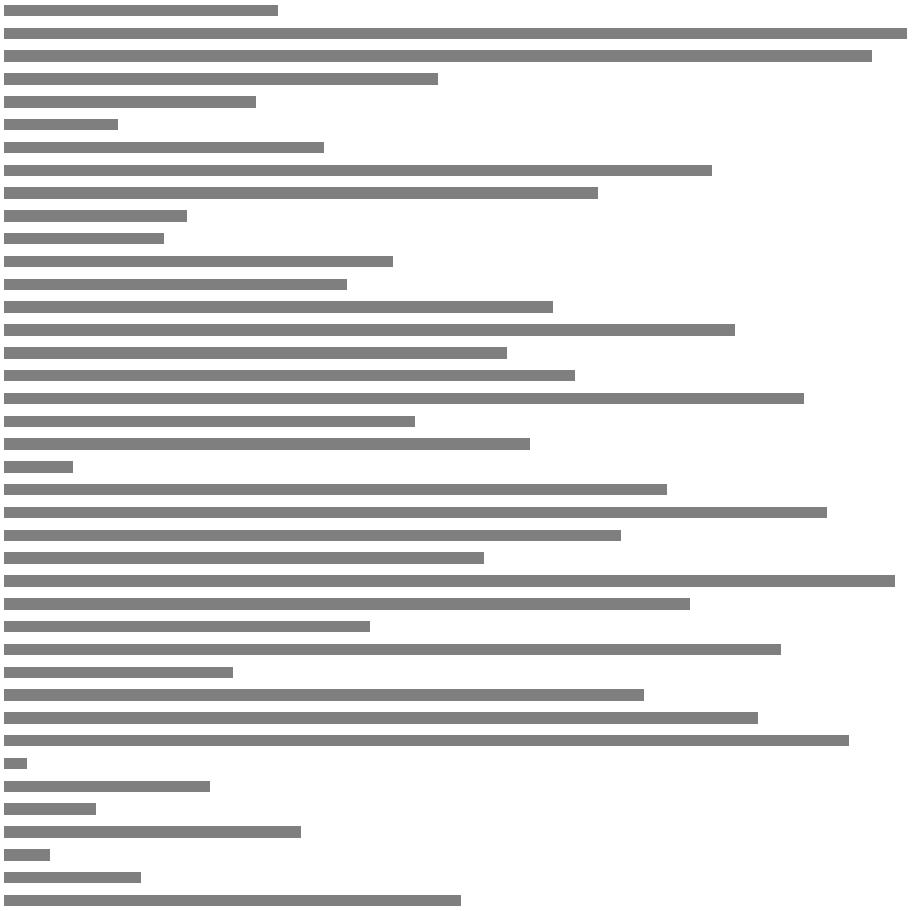
siftDown 4 5 4 7 9 10




swap 5 og 4 4 5 7 9 10


- Animasjon av et enkelt eksempel

# Animasjon:
# In-place heapsort av random data



# Algoritme: In-place heapsort

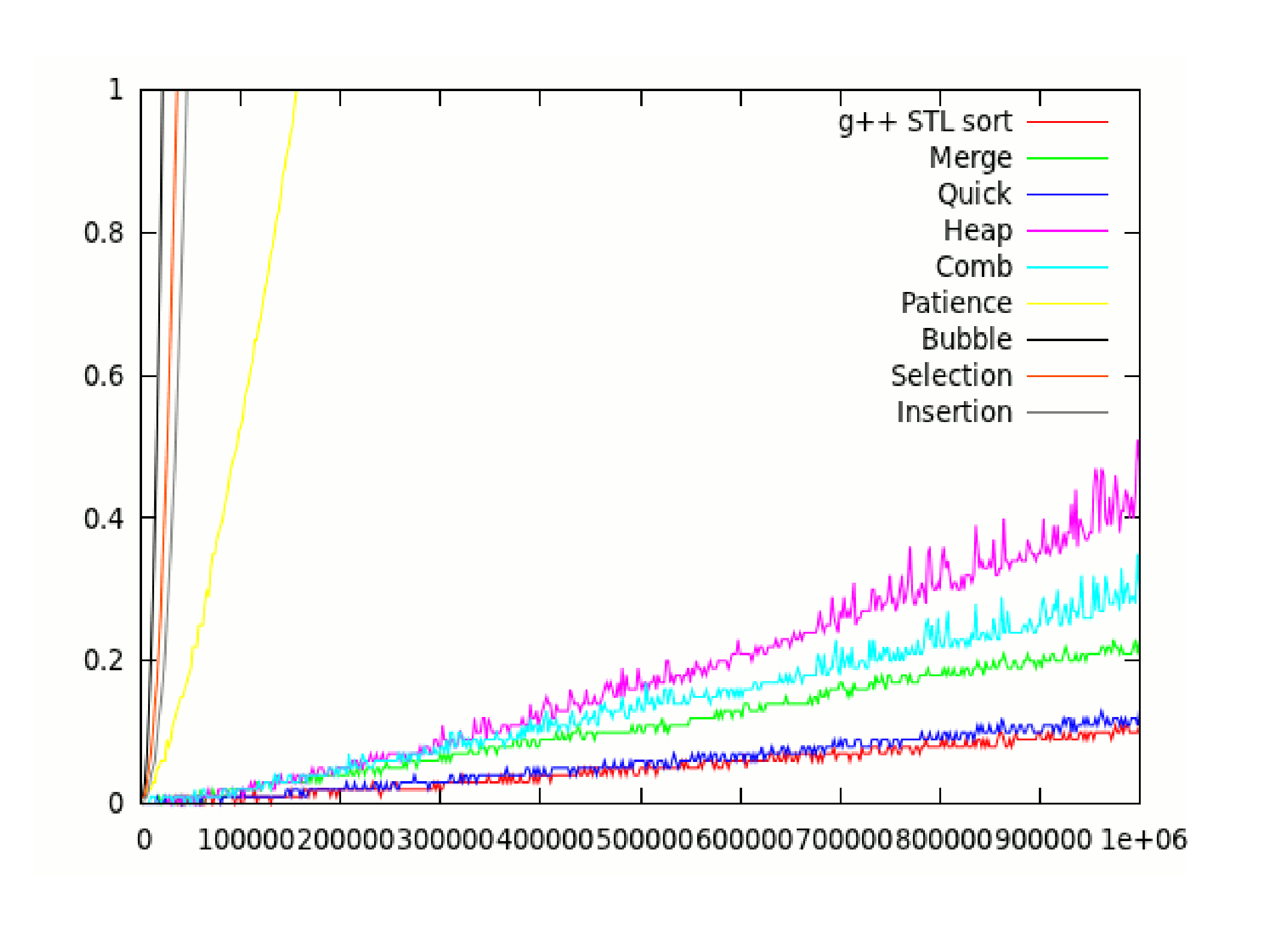
```
void heapSort(int A[])
{
    for (int i = 1; i < A.length; i++)
        siftUp(A, i);

    for (int i = A.length - 1; i > 0; i--)
    {
        int tmp = A[i];
        A[i] = A[0];
        A[0] = tmp;
        siftDown(A, i - 1);
    }
 }
```

# Effektivitet og implementasjon

- $n \cdot$ siftUp + $n \cdot$ siftDown = $O(n \log n)$
- Ikke så rask som Quicksort pga. mye swapping, men:
  - Garantert $O(n \log n)$ uansett data
  - Tilnærmet ingen overhead/ekstra ressursbruk
- Implementasjon: `heapSort.java`
- Testprogram som sammenligner heapsort med Quicksort, mergesort og Javas innebygde sortering av arrayer:

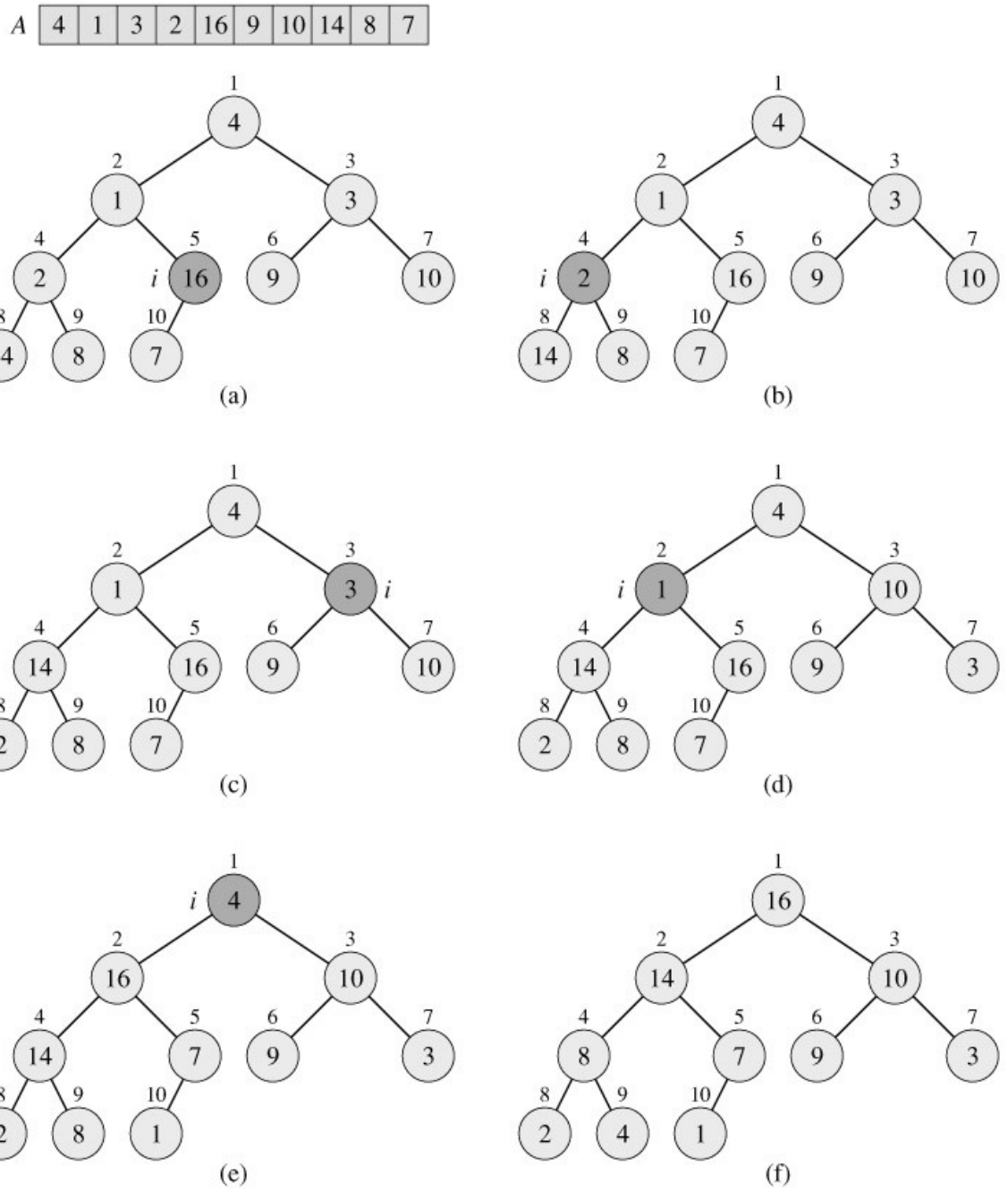
  `testQuickMergeHeap.java` →


g++ STL sort
Merge
Quick
Heap
Comb
Patience
Bubble
Selection
Insertion
1
0.8
0.6
0.4
0.2
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1e+06

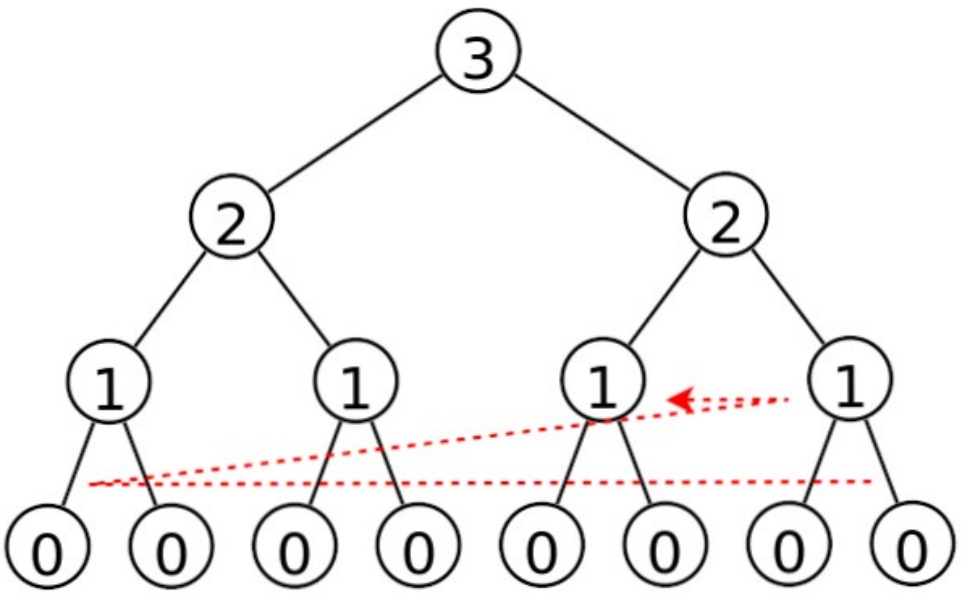
# En variant: Bottom-up heapsort

- Bygger i stedet heapen "fra bunnen av og oppover":
  - Betrakter arrayen som en "heap som må repareres"
  - "Reparerer" først alle delheapene av høyde 1, der rotnoden står på nest nederste nivå, med en siftDown av roten
  - Deretter "repareres" alle delheaper av høyde 2, med rot i nivået over det nest nederste– her står nå kun roten feil så det holder igjen med én siftDown per delheap
  - "Reparasjonen" fortsetter oppover ett nivå av gangen, inntil hele arrayen er omgjort til en heap
- Resten av algoritmen er som standard in-place heapsort
- Kan bevises at "bottom-up" heapsort er litt raskere – oppbygging av heapen blir *O(n)* og ikke *O(n log n)*

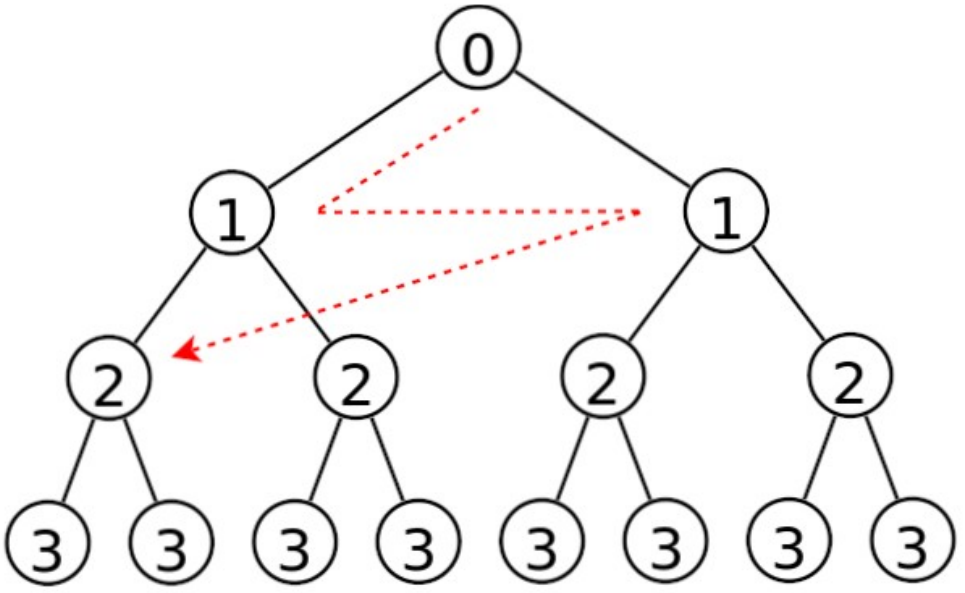
# Bottom-up bygging av heap



# Hvorfor er bottom-up heapsort raskere?




Bottom-up (siftDown)




Top-down (siftUp)

Tallene i hver node angir maksimalt antall flytt av hver verdi ved bygging av heap

- Animasjon av bottom-up-heapsort
- Implementasjon: heapSortBU.java