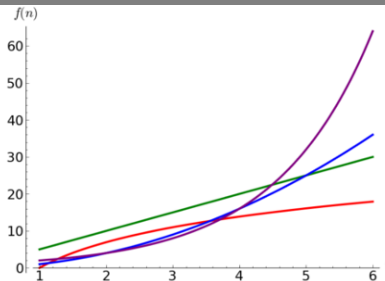


Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 10. Juni 2013

INSTITUT FÜR THEORETISCHE INFORMATIK, PROF. SANDERS



```

function karatsuba(num1, num2)
if (num1 < 10) or (num2 < 10)
    return num1*num2
/* calculates the size of the numbers */
m = max(size(num1), size(num2))
low1, low2 = lower half of num1, num2
high1, high2 = higher half of num1, num2
/* 3 calls made to numbers approximately half the size */
z0 = karatsuba(low1, low2)
z1 = karatsuba((low1+high1), (low2+high2))
z2 = karatsuba(high1, high2)
return (z2*10^(m)) + ((z1-z2-z0) * 10^(m/2)) + (z0)
    
```

- min dauert $O(1)$
- Höhe des Heaps ist $\lfloor \log n \rfloor$
- insert dauert $O(\log n)$
- deleteMin dauert $O(\log n)$

- min dauert $O(1)$
- Höhe des Heaps ist $\lfloor \log n \rfloor$
- insert dauert $O(\log n)$
- deleteMin dauert $O(\log n)$

- min dauert $O(1)$
- Höhe des Heaps ist $\lfloor \log n \rfloor$
- insert dauert $O(\log n)$
- deleteMin dauert $O(\log n)$

- Nacheinander die Elemente dem Heap hinzufügen?
- Korrekte Lösung, aber nicht effizient
- Braucht im Worst-Case $O(n \log n)$ Zeit

- Nacheinander die Elemente dem Heap hinzufügen?
- Korrekte Lösung, aber nicht effizient
- Braucht im Worst-Case $O(n \log n)$ Zeit

```
1 Procedure buildHeapRecursive( $i: \mathbb{N}$ )
2   if  $4i \leq n$  then
3     buildHeapRecursive( $2i$ )
4     buildHeapRecursive( $2i+1$ )
5   siftDown( $i$ )
6
7 Procedure buildHeapBackwards
8   for  $i := \lfloor \frac{n}{2} \rfloor$  downto 1
9     siftDown( $i$ )
```

- n Elemente, $O(\log n)$ im Worst Case pro Element
⇒ $O(n \log n)$ Laufzeit?
- Nein! Höhe der Teilbäume ist "genügend oft" klein
- Damit (Abschätzung siehe VL) ist Laufzeit in $O(n)$

- n Elemente, $O(\log n)$ im Worst Case pro Element
⇒ $O(n \log n)$ Laufzeit?
- Nein! Höhe der Teilbäume ist "genügend oft" klein
- Damit (Abschätzung siehe VL) ist Laufzeit in $O(n)$

```
1 Procedure heapSortDecreasing(a[1..n])  
2   buildHeap(a)  
3   for i:=n downto 2 do  
4     a[i]:=deleteMin
```

- Sortiert absteigend
- $O(n \log n)$ Laufzeit
- Implementierung mit nur $O(1)$ zusätzlichem Speicher möglich
- nicht stabil

- Sortiert absteigend
- $O(n \log n)$ Laufzeit
- Implementierung mit nur $O(1)$ zusätzlichem Speicher möglich
- nicht stabil

- Sortiert absteigend
- $O(n \log n)$ Laufzeit
- Implementierung mit nur $O(1)$ zusätzlichem Speicher möglich
- nicht stabil

- Gegeben sind n Pancakes in unterschiedlicher Größe und gestapelt. Man hat einen Pancake- Flipper zur Verfügung, mit dem man die obersten l Pancakes umdrehen kann (l beliebig). Entwickelt einen schnellen Algorithmus um die Pancakes zu sortieren.
- Spaghettisort

- Gegeben sind n Pancakes in unterschiedlicher Größe und gestapelt. Man hat einen Pancake- Flipper zur Verfügung, mit dem man die obersten l Pancakes umdrehen kann (l beliebig). Entwickelt einen schnellen Algorithmus um die Pancakes zu sortieren.
- Spaghettisort

```
1 Procedure build( $\{e_1, \dots, e_n\}$ )  $M := \{e_1, \dots, e_n\}$ 
2 Function size return  $|M|$ 
3 Procedure insert( $e$ )  $M := M \cup \{e\}$ 
4 Function min return min  $M$ 
5 Function deleteMin  $e := \text{min } M$ ;  $M := M \setminus \{e\}$ ; return  $e$ 
6 Function remove( $h$ :Handle)  $e := h$ ;  $M := M \setminus \{e\}$ ; return  $e$ 
7 Procedure decreaseKey( $h$ :Handle,  $k$ :Key)
8           assert  $\text{key}(h) \geq k$ ;  $\text{key}(h) := k$ 
9 Procedure merge( $M'$ )  $M := M \cup M'$ 
```


Operation	Binary Heap	Fibonacci Heap (Buch)
build	$O(n)$	$O(n)$
size	$O(1)$	$O(1)$
min	$O(1)$	$O(1)$
insert	$O(\log n)$	$O(\log n)$
deleteMin	$O(\log n)$	$O(\log n)$
remove	$O(\log n)$	$O(\log n)$
decreaseKey	$O(\log n)$	$O(1)$ amort.
merge	$O(n)$	$O(1)$