Informatik II

Übung 3

Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz,
Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

Program Today

1. Feedback of last exercise
2. Repetition Theory
3. Next Exercise

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?
  - Start from the bottom. n tries.

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?
  - Start from the bottom. n tries.
- What would be your strategy if you only had two eggs?

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?
  - Start from the bottom. n tries.
- What would be your strategy if you only had two eggs?
  - Use s tries.
  - Use decreasing interval size
  - $s + (s-1) + (s-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \frac{s(s+1)}{2} \geq 100$. Therefore $s = 14$.

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?
  - Start from the bottom. n tries.
- What would be your strategy if you only had two eggs?
  - Use s tries.
  - Use decreasing interval size
  - $s + (s-1) + (s-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \frac{s(s+1)}{2} \geq 100$. Therefore $s = 14$.
  - $\sqrt{n}$

Hottest Path

```
int current = 0;
List<Integer> route = new ArrayList<Integer>();
route.add(0);
while (!food[current]) { // termination criterion
  float max = -1;
  int next = -1;
  for (int j = 0; j < edges.length; ++j) {
    if (edges[current][j] != 0 && max < popularity[current][j]) {
      max = popularity[current][j];
      next = j;
    }
  }
  route.add(next);
  current = next;
}
```

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort				

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$		

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection				

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$		

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion				

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$		

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort				

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$		

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$		

Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

Algorithm Quicksort($A[l, \ldots, r]$)

Input : Array A with length n. $1 \leq l \leq r \leq n$.
Output : Array A, sorted between l and r.

if $l < r$ **then**
    Choose pivot $p \in A[l, \ldots, r]$
    $k \leftarrow$ Partition($A[l, \ldots, r], p$)
    Quicksort($A[l, \ldots, k-1]$)
    Quicksort($A[k+1, \ldots, r]$)

Algorithm recursive 2-way Mergesort(A, l, r)

Input : Array A with length n. $1 \leq l \leq r \leq n$
Output : Array $A[l, \ldots, r]$ sorted.

if $l < r$ **then**
  $m \leftarrow \lfloor (l + r)/2 \rfloor$ // middle position
  Mergesort(A, l, m) // sort lower half
  Mergesort($A, m + 1, r$) // sort higher half
  Merge(A, l, m, r) // Merge subsequences

Algorithm NaturalMergesort(A)

Input : Array A with length $n > 0$
Output : Array A sorted

```
repeat
    r ← 0
    while r < n do
        l ← r + 1
        m ← l; while m < n and A[m+1] ≥ A[m] do m ← m + 1
        if m < n then
            r ← m + 1; while r < n and A[r+1] ≥ A[r] do r ← r + 1
            Merge(A, l, m, r);
        else
            r ← n
until l = 1
```

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don’t change the relative position of two elements.

5	2	6	6	8	4
2	4	5	6	6	8

not stable

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don’t change the relative position of two elements.

5 2 6 6 8 4

2 4 5 6 6 8

not stable

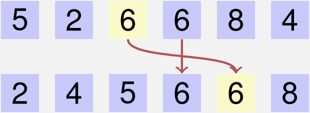
5 2 6 6 8 4

2 4 5 6 6 8

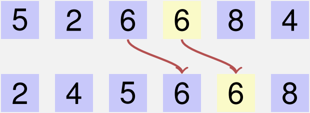
stable

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don’t change the relative position of two elements.



not stable



stable

- In-situ algorithms require only a constant amount of additional memory.

Bonus Exercise

Questions / Suggestions?