Informatik II

Übung 4

FS 2020

Program Today

1. Survey Productive Failure Case Study 1
2. Feedback of last exercise
3. Repetition Theory
   - Analysis of programs
   - Analysis of recurrences
4. Divide & Conquer Sorting Algorithms

3. Repetition Theory

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

The code fragment implies $\Theta(n^2)$ calls to `f()`: the outer loop is executed $n/9$ times and the inner loop contains i calls to `f()`.

Analysis

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Analysis

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`

Analysis

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`
The second inner loop contains $\lfloor \log_2(n) \rfloor + 1$ calls to `f()`. Summing up yields $\Theta(n \log(n))$ calls.

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n/2);
        }
        else{
                f();
        }
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n/2);
        }
        else{
                f();
        }
}
```

There is only a single call to f() at the end of the recursion.

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n-1);
        }
        f();
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n−1);
        }
        f();
}
```

Recurrence

$$T(n) = \begin{cases} T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases}$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n−1);
        }
        f();
}
```

Recurrence

$$T(n) = \begin{cases} T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n)$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Recurrence

$$T(n) = \begin{cases} 2T(n/2) & n > 1 \\ 1 & n = 1 \end{cases}$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Recurrence

$$T(n) = \begin{cases} 2T(n/2) & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n)$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

Recurrence

$$T(n) = \begin{cases} 2T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

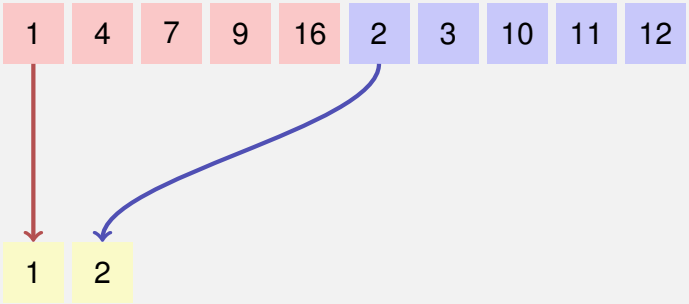
Recurrence

$$T(n) = \begin{cases} 2T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n \log n)$$

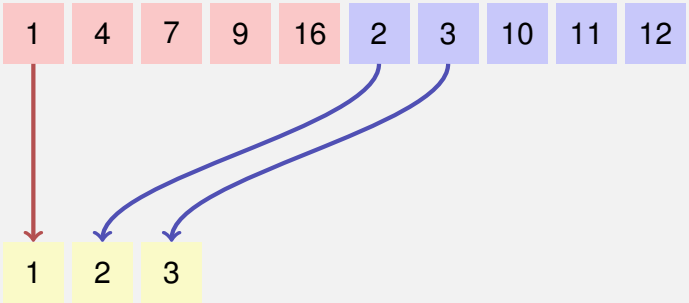
Merge

Merge

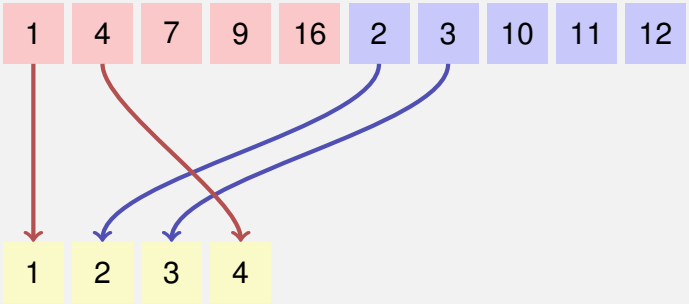
Merge



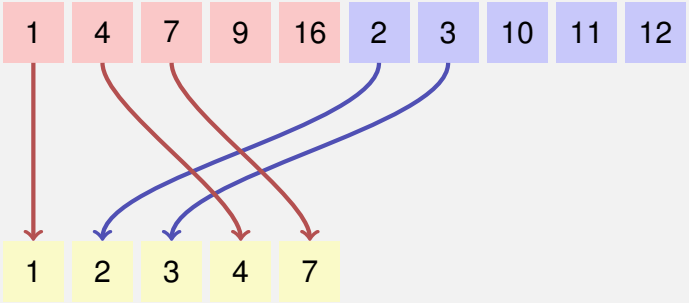
Merge



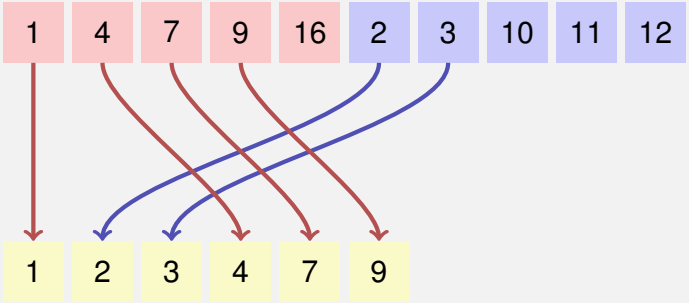
Merge



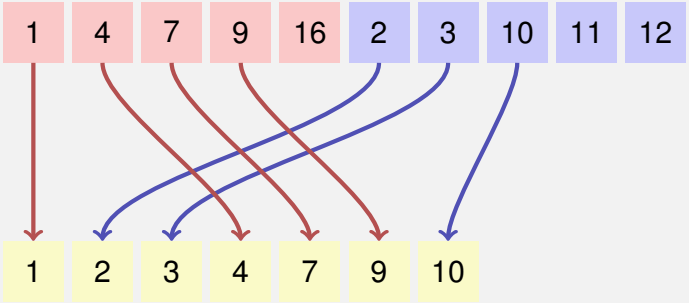
Merge



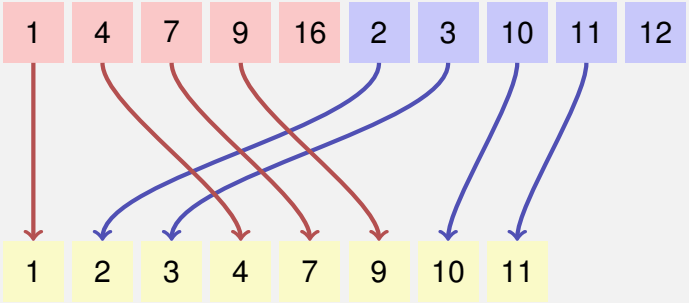
Merge



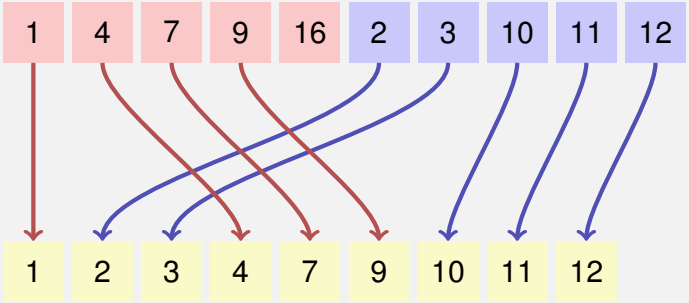
Merge



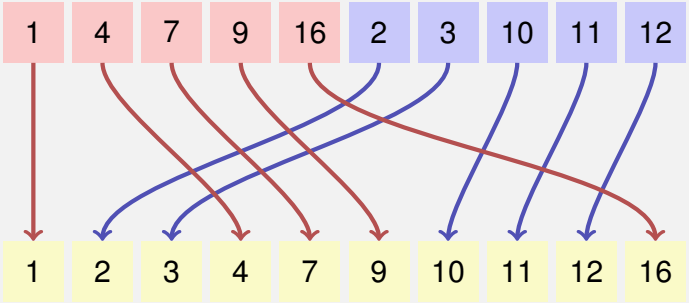
Merge



Merge



Merge



Algorithm `Merge`(A, l, m, r)

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

```
1 B ← new Array(r − l + 1)
2 i ← l; j ← m + 1; k ← 1
3 while i ≤ m and j ≤ r do
4     if A[i] ≤ A[j] then B[k] ← A[i]; i ← i + 1
5     else B[k] ← A[j]; j ← j + 1
6     k ← k + 1;
7 while i ≤ m do B[k] ← A[i]; i ← i + 1; k ← k + 1
8 while j ≤ r do B[k] ← A[j]; j ← j + 1; k ← k + 1
9 for k ← l to r do A[k] ← B[k − l + 1]
```

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

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

if $l < r$ **then**

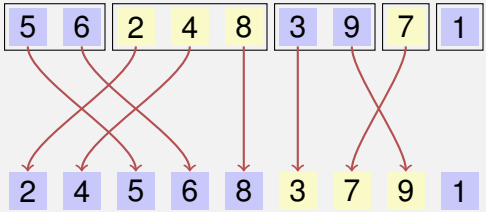
```
m ← ⌊(l + r)/2⌋          // middle position
Mergesort(A, l, m)       // sort lower half
Mergesort(A, m + 1, r)   // sort higher half
Merge(A, l, m, r)        // Merge subsequences
```

Natural 2-way mergesort

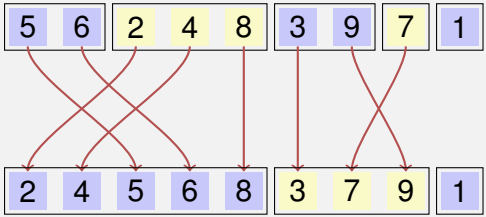
5 6 2 4 8 3 9 7 1

Natural 2-way mergesort

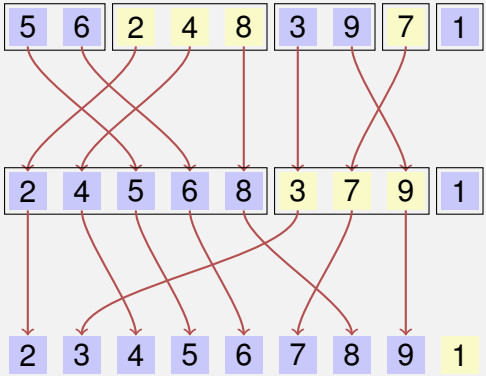
Natural 2-way mergesort



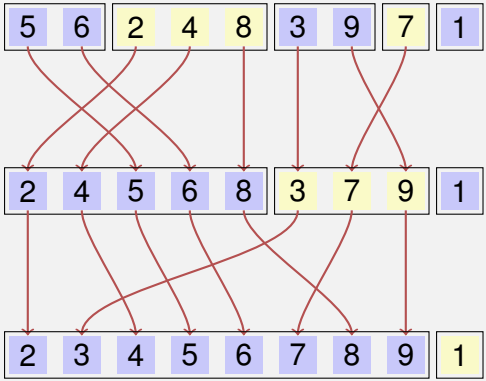
Natural 2-way mergesort



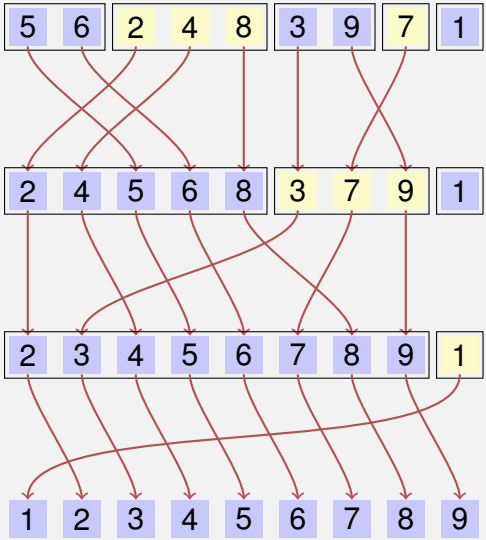
Natural 2-way mergesort



Natural 2-way mergesort



Natural 2-way mergesort



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
```

Algorithm `Partition`(A, l, r, p)

Input: Array A, that contains the pivot p in $A[l, \ldots, r]$ at least once.
Output: Array A partitioned in $[l, \ldots, r]$ around p. Returns position of p.

while $l \leq r$ **do**
    while $A[l] < p$ **do**
        $l \leftarrow l + 1$
    while $A[r] > p$ **do**
        $r \leftarrow r - 1$
    swap($A[l]$, $A[r]$)
    if $A[l] = A[r]$ **then**
        $l \leftarrow l + 1$

return l-1

Illustration Partition

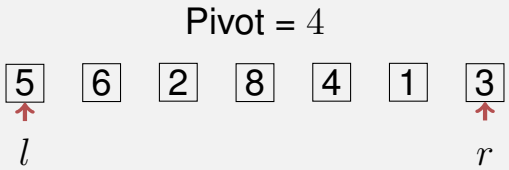


Illustration Partition

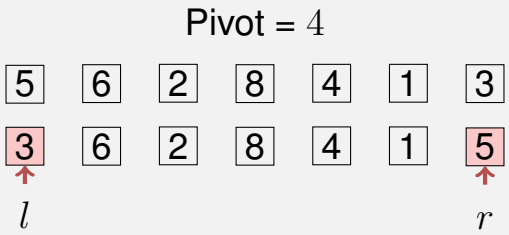


Illustration Partition

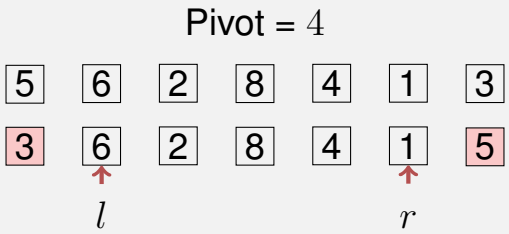


Illustration Partition

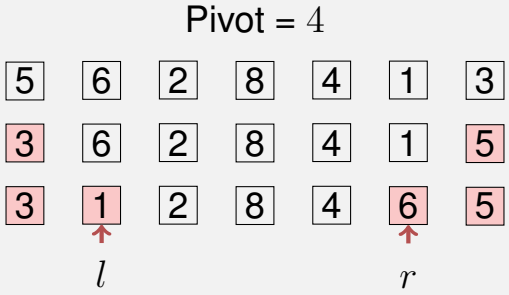


Illustration Partition

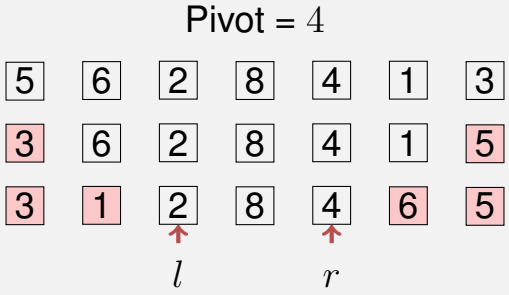


Illustration Partition

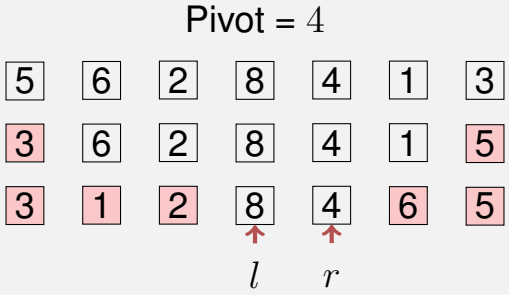


Illustration Partition

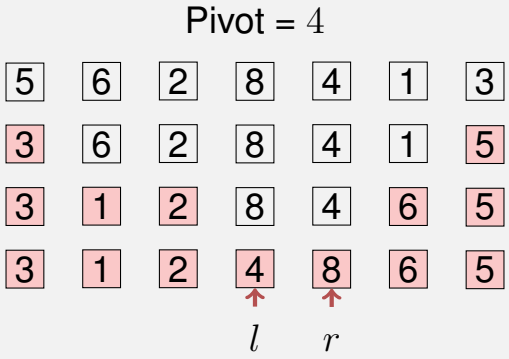
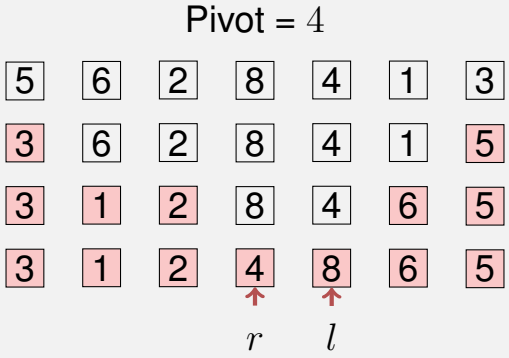


Illustration Partition

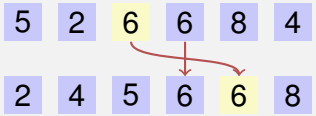


Algorithm `Quicksort`(A, l, r)

Input: Array A with length n. $1 \leq l \leq r \leq n$.
Output: Array A, sorted in $A[l, \ldots, r]$.
if $l < r$ **then**
    Choose pivot $p \in A[l, \ldots, r]$
    $k \leftarrow$ `Partition`(A, l, r, p)
    `Quicksort`$(A, l, k-1)$
    `Quicksort`$(A, k+1, r)$

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don't change the relative position of two elements. [1]

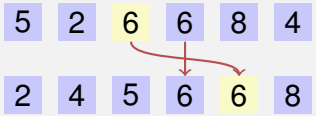


not stable

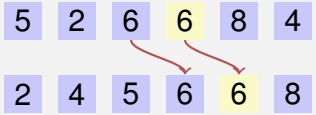
[1] Every sorting algorithm can be made stable by storing the array position together with the element.

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don't change the relative position of two elements. [1]



not stable

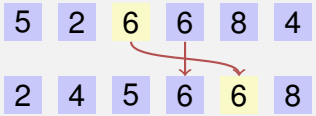


stable

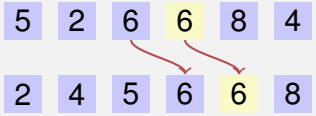
[1] Every sorting algorithm can be made stable by storing the array position together with the element.

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don't change the relative position of two elements. [1]



not stable



stable

- In-situ algorithms require only a constant amount of additional memory. (Mergesort is not in-situ)

[1] Every sorting algorithm can be made stable by storing the array position together with the element.

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort				

Quiz: Sorting and Running Times

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

Quiz: 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				

Quiz: 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	$\Theta(n^2)$	$\Theta(n^2)$		

Quiz: 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				

Quiz: 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	$\Theta(n \log n)$	$\Theta(n \log n)$		

Quiz: 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				

Quiz: 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	$\Theta(n \log n)$	$\Theta(n^2)$		

Quiz: 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	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$*
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$		

Quiz: 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	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$*
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

* non-trivial (and not derived in class)

Questions / Suggestions?