Data structures lab – week 9

Welcome back!

Agenda for today

- Final word on assignment 4
- Sorting
- Divide and conquer
- How to speed things up
- Assignment 5

Wake-up quiz – assignment 4

- What is the key for first hidden message?
  - a) 1
  - b) 2
  - c) 3
  - d) 4

Wake-up quiz – checking progress

- What is the key for first hidden message?
  a) 1
  b) 2
  c) 3
  d) 4
- The correct answer is...
  - I won't tell you if you don't know
    - I'm just checking in

Assignment 4 – decrypting

- Remember, implement extract-min
  - Do not use heapsort to do your job
    - Tempting to do though
      - One build-heap
      - One heapsort
      - Look at every (*k+1)*th word
    - I basically already gave you all the code to do this. So don't :-)
  - Even your program from assignment 3 could be used to decrypt the messages
    - But you need to learn about heaps.

Increasing array size

- Someone wondered how real-world implementations deal with array increases.
  - Java's `ArrayList` source code:
    - `newCapacity = (oldCapacity * 3)/2 + 1`
  - After that, it is native system calls (C code)
    - System.copyarray()
    - Makes a so-called “shallow” copy
      - Does not create new objects, only copied references
      - According to online forums

Cryptographic systems

- Someone else wondered:
    - “Isn't this called a symmetric cipher, since both sender and recipient have the same key? A public-key system would be a more secure but it's more difficult to implement.”
  - Yes, it is a symmetric key cryptographic system you are implementing.
  - But a public-key system is not necessarily more secure. They both have advantages and disadvantages
    - But let's not pick up that discussion now

Sorting

- Sorting is a fundamental operation
- Sorting is a sub-routine in many algorithms
  - Shortest path
  - Scheduling
  - Computational geometry
  - Many more

Sorting – complexity

- Sorting has a proved lower bound of $\Omega(n \lg n)$ for comparison sorts.
  - Comparison meaning comparing elements
- If certain (true) assumptions are made, the running time can be reduced to linear time in some cases
  - But read chapter 8 if you want to know more about that

Sorting – implementation

- You have already implemented data structures that support fast O(nlgn) sorting
  - Heaps
    - Heapsort
  - Binary search trees (balanced)
    - In-order-tree-walk
      - Not usually used for sorting

Assignment 5

- Optional
  - But only if you have more than 380 points!
- Due one week from now
- Implement quicksort
- Implement at least two other sorting algorithms
- Compare performance
  - Small write-up, ½-1 page, maybe with a graph

Divide and conquer – the concept

- Divide: Split a problem into subproblems
- Conquer: Solve each subproblem recursively
- Combine: Combine the solutions

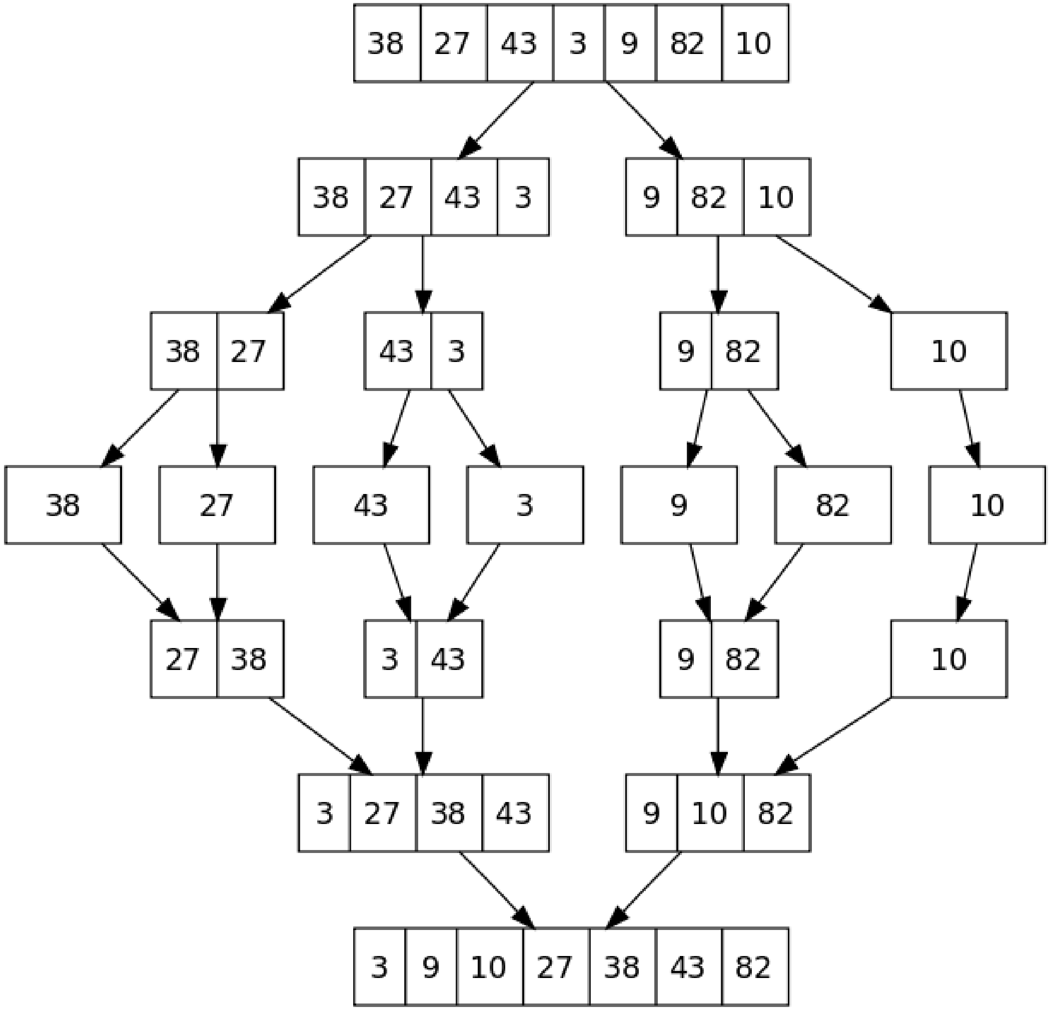
Merge-sort

- Is a divide and conquer algorithm
- $\Theta(n \lg n)$ running time.
- Simple implementation

Merge-sort – algorithm

- For an array *A* with *n* elements
  - Divide: Create subarrays of size *n*/2
    - Until reaching a base case where the sub arrays have length 1
  - Conquer: Sort the subarrays recursively
  - Combine: Merge the sorted subarrays

Merge-sort



38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Wake-up quiz – merge-sort

- For an array of size n, what is the running time of the *merge* procedure?

    a) O(1)

    b) O(lg n)

    c) O(n)

Wake-up quiz – merge-sort

- For an array of size n, what is the running time of the *merge* procedure?
  - a) O(1)
  - b) O(lg n)
  - c) O(n)
- The correct answer is c
  - That must mean that the array is divided lg(n) times
    - But you already knew that because of your knowledge with binary search trees.

Quicksort

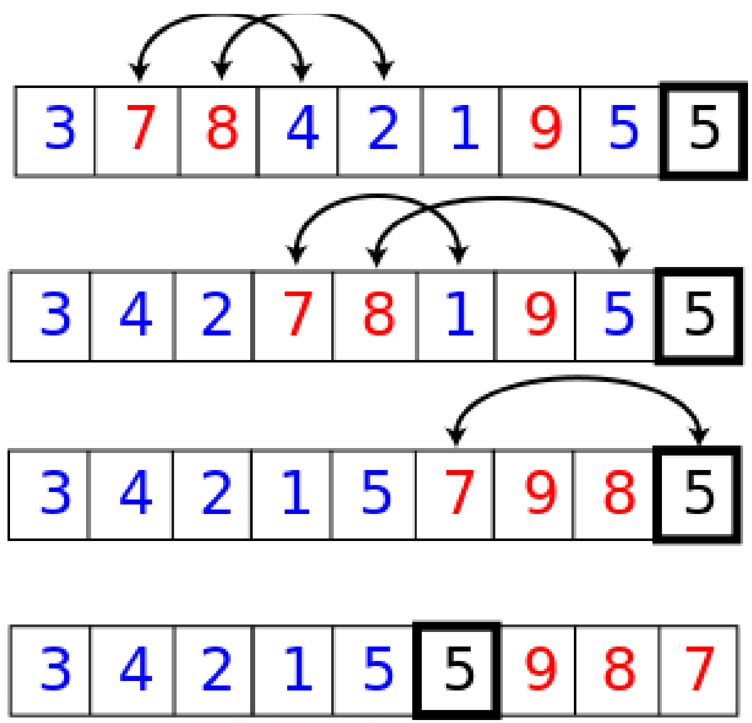
- Is a divide and conquer algorithm
- $O(n \lg n)$ average case running time but $O(n^2)$ worst case running time
  - Worst case rarely happens
- Widely used for sorting
  - Java's Arrays.sort uses the quicksort
    - But not from CLRS though
  - Probably also C++ but I couldn't find confirmation for this.

Quicksort – algorithm

- For an array *A* with *n* elements
  - Divide: Partition *A* into two subarrays around an index *q* such that
    - Values of elements A[0..*q*-1] are less than (or equal to) A[*q*]
    - Values of elements A[*q*+1,*n*-1] are greater than (or equal to) A[*q*]
  - Conquer: Recursively sort the the subarrays
  - Combine: The subarrays are already sorted so we do not need to combine.

Quicksort – partition

- 5 is the *pivot* element
- We maintain pointers to current element less than and greater than 5



Wake-up quiz – quicksort

- For an array of size n, what is the running time of the *partition* procedure?

    a) O(1)

    b) O(lg n)

    c) O(n)

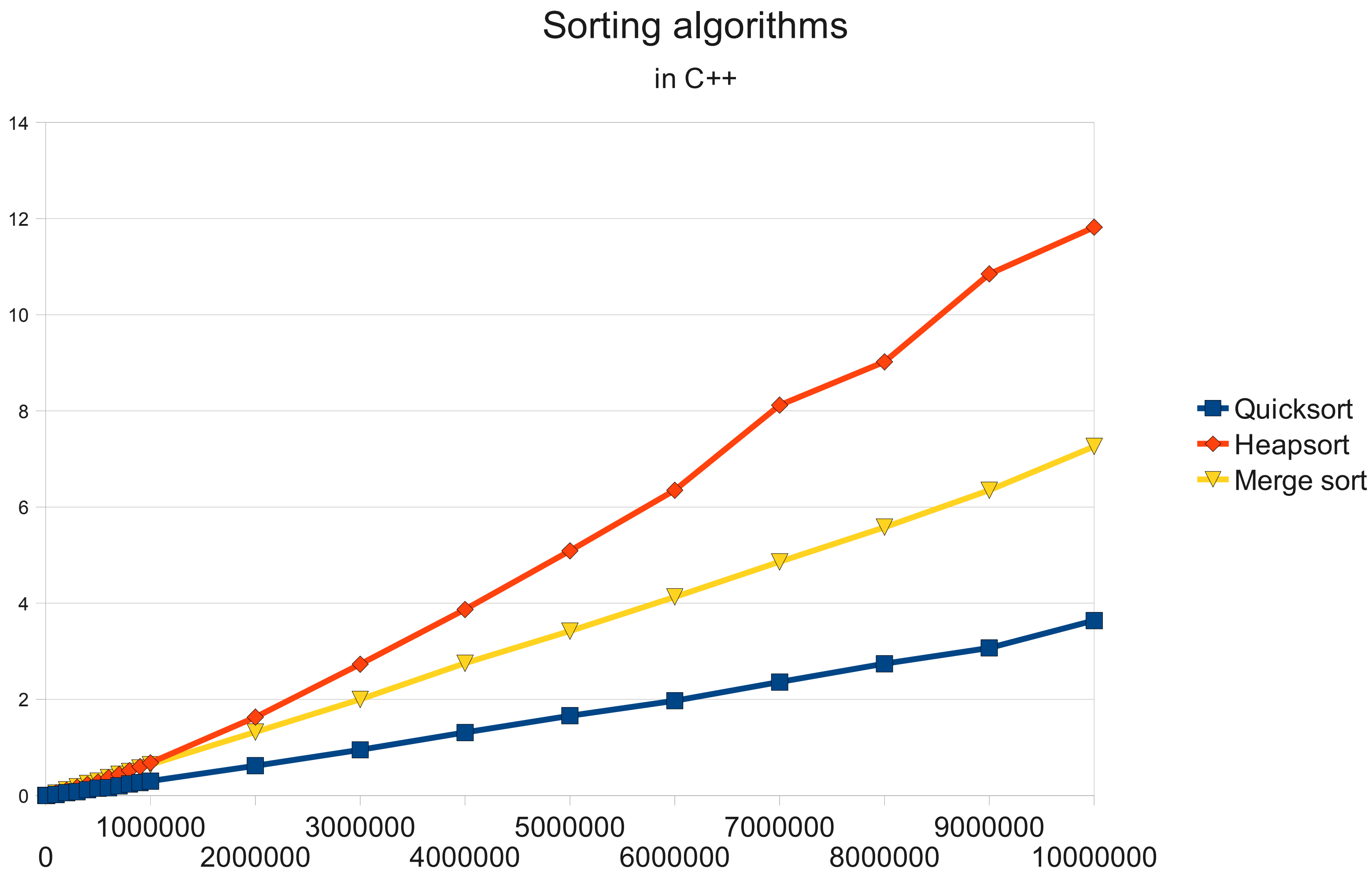
Wake-up quiz – quicksort

- For an array of size n, what is the running time of the *partition* procedure?
  - a) O(1)
  - b) O(lg n)
  - c) O(n)
- The correct answer is c
  - That must mean that we *hope* to split the array lg(n) times
    - Depends on the pivot
    - Hints why we have $O(n^2)$ worst case

Sorting – comparison

- I implemented heapsort, mergesort and quicksort in C++
  - Similar to assignment 5
- I did not use fancy data structures
  - Just plain old `int` arrays
- Input sizes from 100-10,000,000
- Which one do you think is the fastest?

Sorting – results



Sorting – conclusion

- Quicksort is fastest
- But maybe we can do better than this?
  - Without changing algorithms

Speeding things up

- Merge-sort and quicksort are cool
  - They are both divide-and-conquer
  - They have the same average case running time
    - They are fast
  - They are easy to make faster
    - In theory at least
- Let's talk concurrency a bit

Parallelism / Concurrency

- Parallel and concurrent is not the same
- Sun's “multithreaded programming guide”
  - Parallelism:
    - “A condition that arises when at least two threads are executing simultaneously.”
  - Concurrency
    - “A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism”

Parallelism / Concurrency

- For two processes P1, P2:
  - Parallelism:
    - P1 and P2 can execute at the exact same time
    - E.g. executing P1 and P2 on separate CPUs.
  - Concurrency:
    - P1 and P2 may overlap in their execution
      - But not necessarily run in parallel
    - E.g. executing P1 and P2 on the same CPU (multitasking)

Concurrent programming

- Most modern programming languages have support for concurrency
  - Java: `Thread` in java.lang
    - Easy to use
    - I'll focus on this one
  - C++: pthread
    - Not so easy to use
  - Python: `Thread` in threading
    - I haven't played with it

Why concurrency?

- Operating systems would not work without multitasking
- Large-scale database systems would not work without concurrency
- Games would be unplayable without concurrency
- Because it can speed things up

Concurrency in Java

- Threads of the class `Thread` can run concurrently
  - Not necessarily in parallel.
- Each thread runs a small subprogram that is of the type `Runnable`
- Implement either interface `Runnable` or extends class `Thread`.

Concurrency in Java – example

- A class that just prints 0 to 9

```
public class PrintTenNumbers
implements Runnable {
  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println(i);
    }
  }
}
```

Concurrency in Java – example

- Making two threads that does this

```
public static void main(String[] args)
{
   Thread t1 = new Thread(new
PrintTenNumbers());
   Thread t2 = new Thread(new
PrintTenNumbers());
   t1.start(); // Starts the thread
   t2.start();
   t1.join(); // Wait for thread to stop
   t2.join();
}
```

Concurrency in Java

- Theoretically, the previous example should print out 0 to 9 in any order, interleaving between the threads
- In practice, this is not always the case
  - Java does not allow you to control that you want something executing on different CPUs / cores, i.e. true parallelization
    - So we can only assume that they do

Speeding things up – merge-sort

- Merge-sort recursively calls itself on equal sized subarrays that are distinct
  - Easy to parallelize
    - Solve subproblems in separate threads of execution
  - The merge procedure of two subproblems cannot be parallelized

Speeding things up – merge-sort

- Non-parallel version

```
public void mergeSort() {
  sort(0,A.length-1);
}

private void sort(int p, int r) {
  if (p < r) {
    int q = (p+r)/2;
    sort(p,q);
    sort(q+1,r);
    merge(p,q,r);
  }
}
```

Concurrent merge sort

- Parallel merge sort

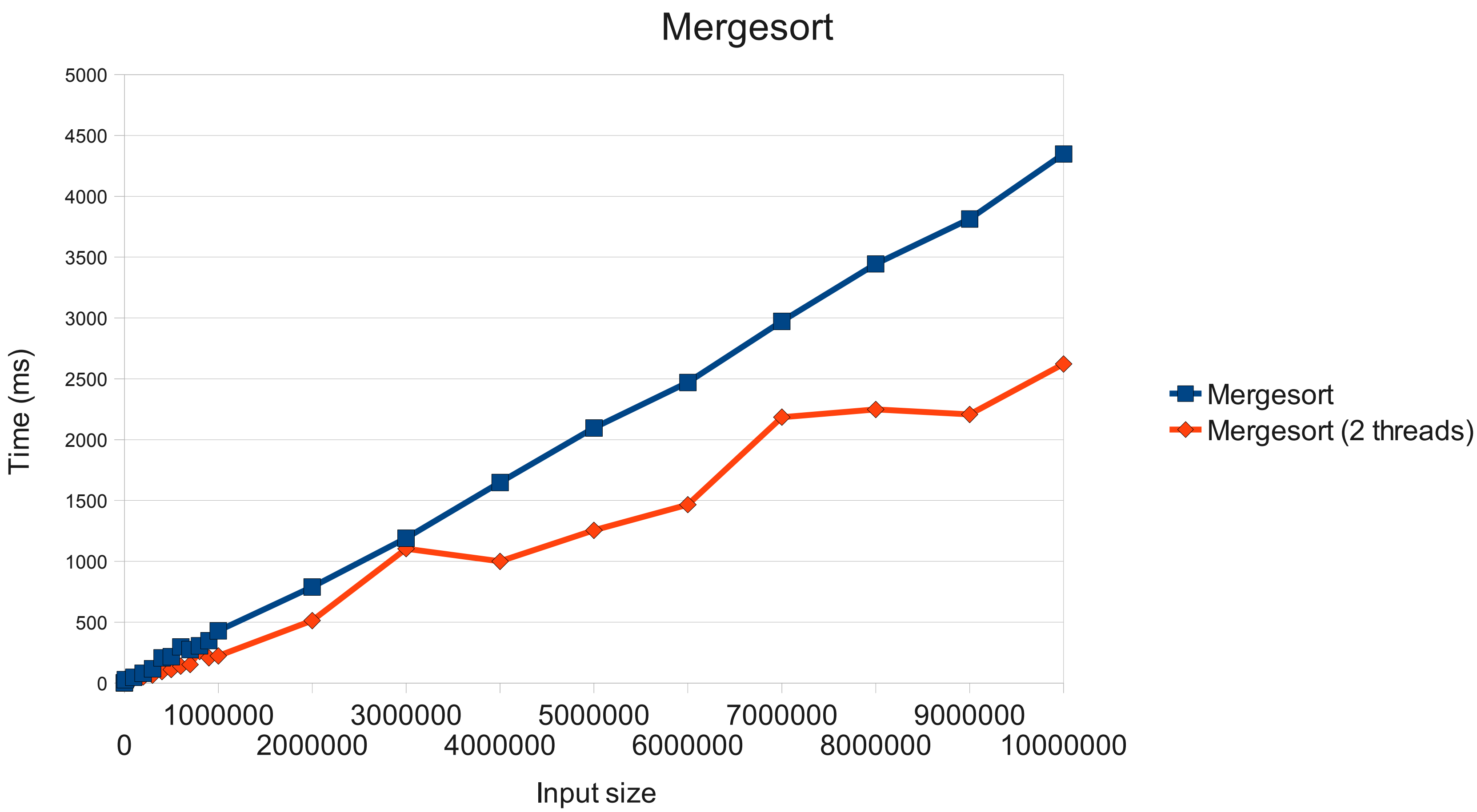
```
public void parallelMergeSort() {
   final int q = A.length/2-1;
   // Declare threads t1 and t2
   // See next slide
   t1.start();
   t2.start();
   t1.join();
   t2.join();
   merge(0, q, A.length-1);
}
```

Concurrent merge sort

```
Thread t1 = new Thread(new Runnable() {
  @Override
  public void run() {
    sort(0,q);
  }
});
```

- Same for t2 but with:
  `sort(q+1,A.length-1);`
- On previous slide, “join” needs to be surrounded with try-catch

Concurrent merge sort – results



Concurrent merge sort – conclusion

- Running merge sort concurrently with only two threads speeds up execution
  - It is consistently faster
    - It is not consistently improving speed
- I cannot verify if it actually runs on both cores of my system

Speeding things up – quicksort

- We can speed up quicksort like merge sort
- The partition procedure cannot be easily parallelized (just like merge)
- Also, subproblems are not necessarily equal in size
    - Because of pivot element
        - Potentially no speed up

Speeding things up – quicksort

- Non-parallel quicksort

```
public void quickSort() {
   sort(0,A.length-1);
}

private void sort(int p, int r) {
   if (p < r) {
      int q = partition(p,r);
      sort(p,q-1);
      sort(q+1,r);
   }
}
```

Concurrent quicksort

- Parallel quicksort

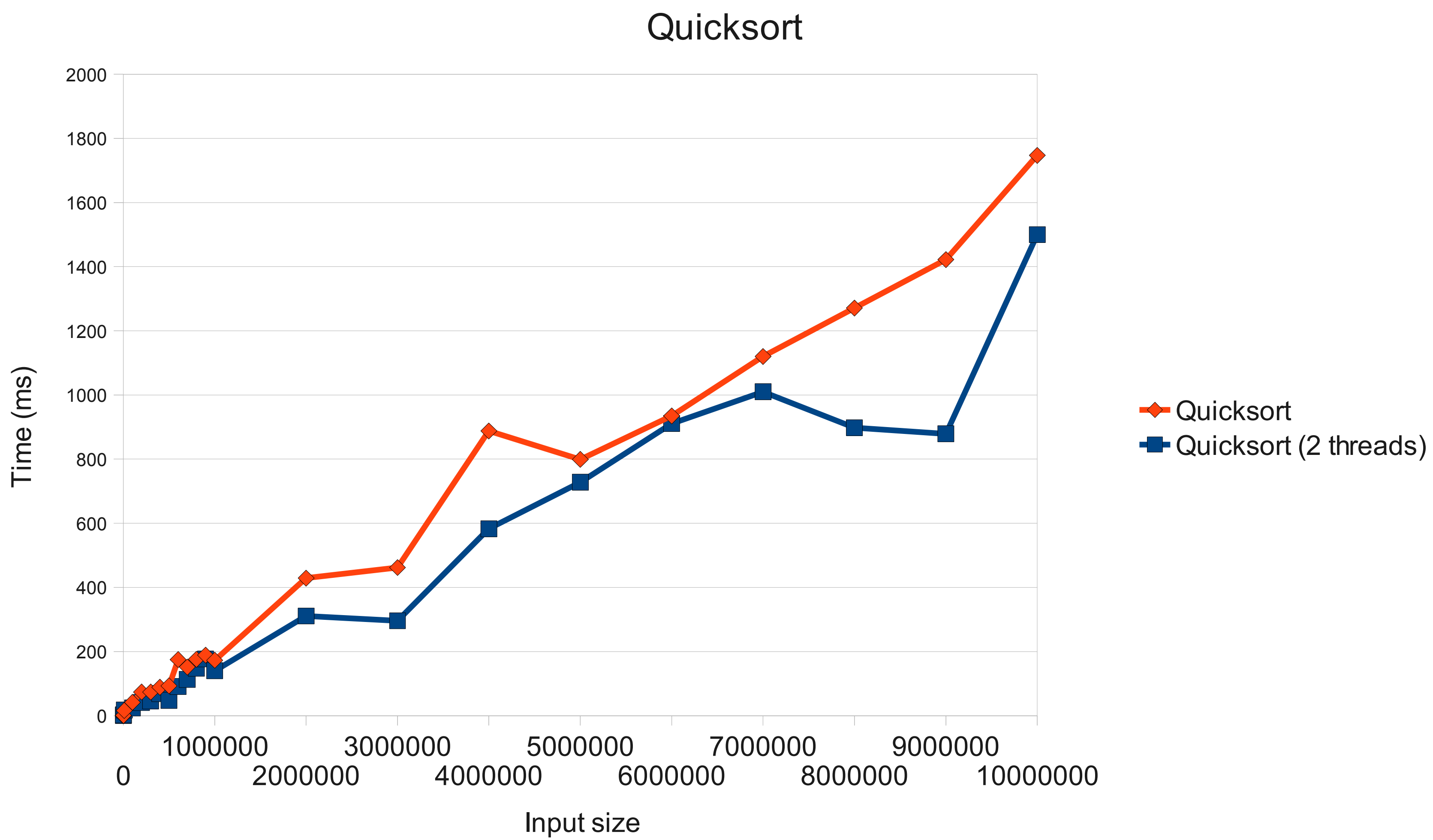
```
final int q = partition(0, A.length-1);
  // Declare threads t1 and t2
  // See next slide
  t1.start();
  t2.start();
  t1.join();
  t2.join();
}
```

Concurrent quicksort

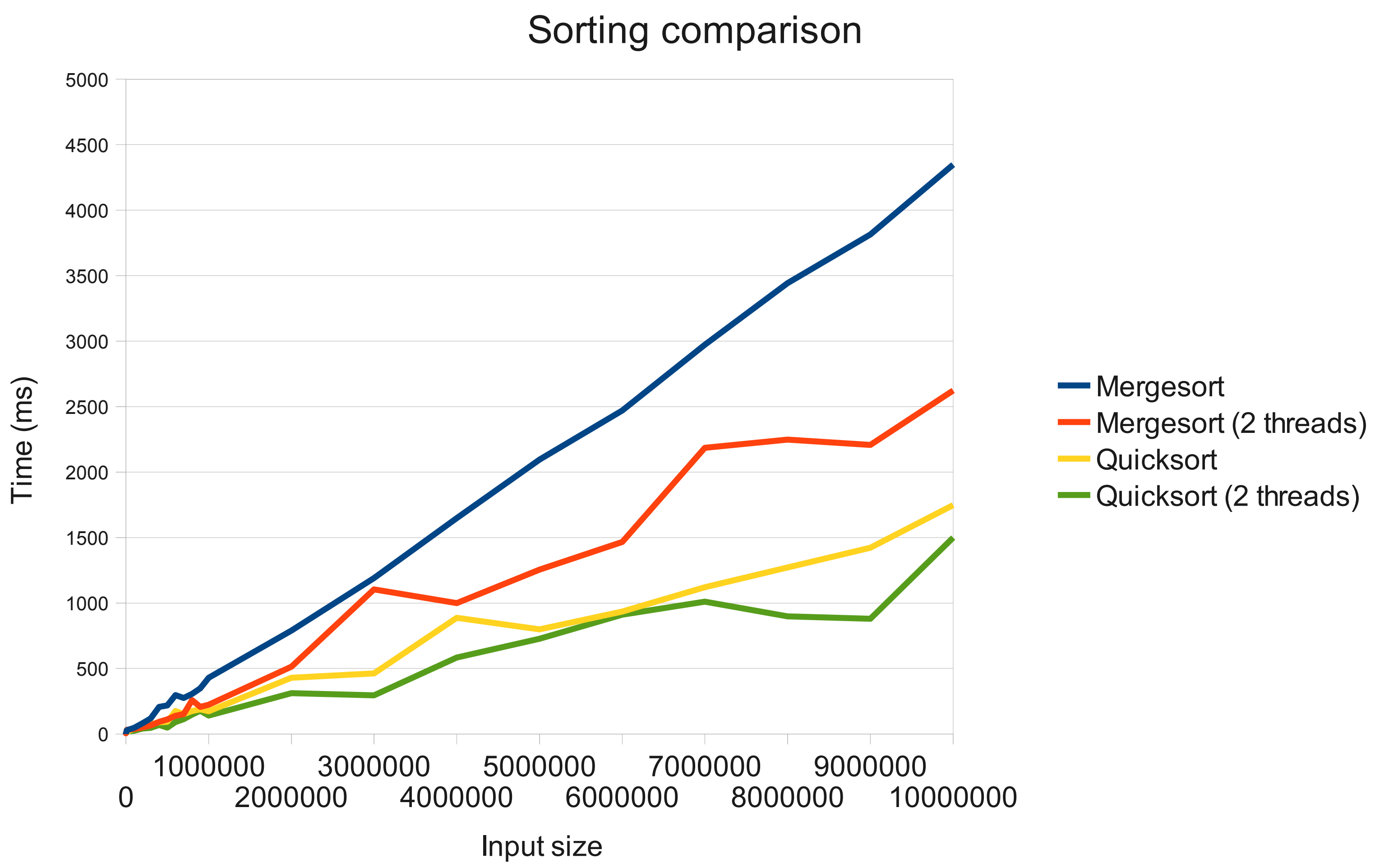
```
Thread t1 = new Thread(new Runnable() {
  @Override
  public void run() {
    sort(0,q-1);
  }
});
```

- Same for t2 but with:
  `sort(q+1,A.length-1);`
- On previous slide, “join” needs to be surrounded with try-catch

Concurrent quicksort – results



Sorting – results



Are we done?

- Class democracy
  - Should we have a class next week?

Class summary

- You (should) have
  - Implemented important data structures
  - Learned (somewhat) to use C++
  - Got an understanding of how to go from an abstract description (pseudocode) to concrete implementation
  - Learned to solve problems largely by yourself or with small hints
  - Had some fun with it all

Class summary

- And that's it, I guess.
- There's no final exam
- Good luck with assignment 5
- Good luck next term

Thank you

Questions?