

Flipped Lecture 03 – 09.25.14

Questions worth asking

Here is a set of questions and possible answers for your reference.

1. *Why should I ever use selection sort, when there is insertion sort?*
 - a. **Answer:** True. Only good thing about selection sort is it only does one exchange. Still it does a lot of comparisons (to find min) and that can be costly too.
2. *Should I use insertion sort if I am not sure about the data set?*
 - a. **Answer:** Probably not. If you know something about the data set, like it is small ($n < 100$), almost sorted, insertion sort is a good choice. Insertion sort with few optimizing tricks are frequently used in applications.
3. *Recursion seems to be a big overhead in mergesort. So when is it better to apply mergesort?*
 - a. **Answer:** If you need a consistent linearithmic algorithm that performs equally well regardless of the data set. For example Java 6 uses some variation of mergesort as the system sort. As for recursion, it is an overhead, but computers have figured out a way to optimize things
4. *Is it possible to combine two sorts like mergesort and insertion sort?*
 - a. **Answer:** Yes, this is frequently done in practice. You can stop recursive divisions of mergesort when the subarray becomes size < 100 for example. So one can write a code like: if ($hi-lo < 100$) insertionSort inside the mergesort code.
5. *Why is it that $A[i]$ is easier to access than $A[i+4096]$ for example?*
 - a. **Answer:** This has to do with how computers fetch memory from the RAM. Most likely you have accessed $A[i-1]$ (or $A[i+1]$) and now trying to access $A[i]$. It is very likely that $A[i]$ is immediately available in cache (closer to CPU). But $A[i+4096]$ may not be readily available and one must make a trip to RAM to fetch it.
6. *What is the deal with random function in computers? Someone told me it is really a pseudo random number*
 - a. **Answer:** True. Computers are deterministic machines, that calculates the next one based on previous one. But in true random numbers all calculations must be mutually independent. So technically if you can figure out the starting value of a computer generated random number sequence, you most likely can correctly guess the next values of the sequence. So it is not random at all. For true randomness, look to nature (may be that is not random either. Some higher power knows how nature works)
7. *How do we know Knuth shuffle produces a uniformly distributed set of random numbers.*
 - a. **Answer:** It can be shown. See the results from 1938 proposition.
http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

8. *why do we need comparable and comparators?*

- a. **Answer:** You can make a collection implements comparable so it will tell you how to compare two elements by using compareTo method. On the other hand , if a class is implementing Comparable, then you can have multiple comparator objects that can define different orders on the collection

9. *Why do we need to have stable sorts?*

- a. **Answer:** Stable sorts maintain relative order of equal keys. So if one needs to sort a set by last name and then first name, it will maintain the last name order for equal last names while sorting the first name.

10. *What should I look for in a stable sort?*

- a. **Answer:** basically I like to think that any algorithm with sliding property (no exchanges) seems to be a good candidate for stable sorts. We know insertion sort and mergesort are stable as they seem to follow this property.