Insertion Sort

Insertion Sort

Lets Look at the Code:

```
for j = 2 to A.length
    key = A[j]
    // insert A[j] into the sorted sequence A[1...j-1]
    i = j-1
    while i>0 and A[i] > key{
        a[i+1] = A[i]
        i = i-1
    }
    a[i+1] = key
```

How to show an algorithim is correct

Show Three Properties of invariant:

1. **Initialization**: the invariant is true before 1st iteration
2. **Maintenance**: if invariant is true, before iterition it is true, after iteration i
3. **Termination**: at the end invariant gives some useful property that shows algorithm correctness

Proof for **insertion sort** is:

1. A[0] is sorted
2. At iteration j *(line 2)*
    1. We retrieve A[j] and linearly search A[i...j-1] for A[j]'s sorted position *(line 5)*
    2. insert into sorted pos *(line 8)*
    3. A[1...j] will be sorted
3. if j>n we terminate *(line 1)*
    1. Because each loop increases j by 1, we must ahve j=n+1
    2. Substitute into invariant A[1...n] is sorted *(proof)*

Runtime for **insertion sort** is:
$T(n) = \Sigma_{linei} cost(i) * \#timesran(i)$
$T(n) = c_1 * n + c_2(n-1) + c_4(n-1) + c_5\Sigma_{j=2}tj + c_6\Sigma_{j=2}(t_j - 1) + c_7\Sigma_{j=2}(t_j - 1) + c_8(n-1)$
Cases:

- Best Case - Shortest running time for every input
- Worst Case - Slowest Running time for any input
- Average Case - Can be calculated depending on your conditions
  - Distrobution assumptions about Data (certain algorithms do better in certain cases)
  - Algorithm Makes Random Choices (quicksort)

Case for **Insertion sort**:
$\Sigma_{j=2} j$ = $\Sigma_{j=1}(j-1)$ = $\frac{n(n+1)}{2} - 1$ = $O(n^2)$

An Algoithim for finding the k'th Percentile

First we must sort, then we must pull the center value.
Best Sorting Alg, O(nlog(n))
Mergesort:
$T(n) = 2T(\frac{n}{2}) + O(n)$
Option 1: reduce the work per subproblem and increase the number of subproblems
$T(n) = 2T(\frac{n}{2}) + O(1)$
Option 2: Do more work per problem on less subproblems
$T(n) = T(\frac{n}{2}) + O(n)$

Problem: Selection

- Go through each item in the list
- Pick a random point i
- Seperate them into two lists, one with one containing all the elements greater than L[i] and one with all the elements less than L[i]
  - We now run this again on the largest list
  - we cut our problems in half with a runtime of O(n)
- return median

Time:

- Worst case: $O(n^2)$
- Best Case: $O(n)$
- Average Case:
  - Lets assume we find a pivot thats "good enough", where good enough means between 25th and 75th Percentile
  - Worst case here would be 3/4n (we pick on 25th or 75th)
  - $E[T(n)] = E[T(\frac{3}{4}n) + O(n)]$
  - E = Expected number of tosses of a fair coin before heads observed

    - E= 1+.5+.25+.....=2
    - $Pr(x = k) = (1-p)^{k-1}$

Similar to Induction:

1. **Basecase**
2. **Hypothesis**
3. **Conclusion**