Week 5: Searching and Sorting

Searching and sorting seems like such primitive tasks that everything must have all been said and done about them already, right? Actually not. They are so fundamental that people keep inventing newer and better ways to use them in new and special ways. We can only scratch the surface of them this week, but scratch it we will!

Exercises

Binary search

We argued that the worst-case running time for the binary search is $O(\log n)$.

Exercise: What is the best-case running time, and what would the input data look like to achieve it?

Selection sort

Exercise: Give an example input where selection sort is not stable.

Insertion sort

We argued that the worst-case running time for insertion sort was $O(n^2)$ but the best-case running time was $O(n)$.

Exercise: Describe what the input data, `numbers`, should look like to actually achieve the worst- and best-case running times.

Bubble sort

Recall the invariants of the inner (I) and outer ($O_\#$) loop of bubble sort from the book:

$$I : \forall k \in [0, i-1) : x[k] \leq x[i-1]$$
$$O_1 : \forall k \in [n-j, n-1) : x[k] \leq x[k+1]$$
$$O_2 : \forall k \in [0, n-j) : x[k] \leq x[n-j]$$

Exercise: Since O_1 and O_2 tells us that the last j elements are already the largest numbers and are already sorted, we do not need to have the inner loop iterate through these last j elements. How would you exploit this to improve the running time of bubble sort? The worst-case behaviour will not improve, but you can change the running time to about half of the one we have above. Show that this is the case.

Exercise: With cocktail sort, after running the outer loop j times, both the first j and the last j elements are in their final positions. Show that this is the case.

Exercise: Knowing that both the first and last j elements are already in their right position can be used to iterate over fewer elements in the inner loops. Modify the algorithm to exploit this. The worst case complexity will still be $O(n^2)$, but you will make fewer comparisons. How much do you reduce the number of comparisons by?

Comparison sort comparison

Exercise: Insertion sort runs in $O(n^2)$ when the input is sorted in the reverse order, but can process sorted sequences in $O(n)$. If we can recognise that the input is ordered in reverse, we could first reverse the sequence and then run the insertion sort. Show that we can reverse a sequence, in place, in $O(n)$. Try to adapt insertion sort, so you first recognise consecutive runs of non-increasing elements, then reverse these before you run insertion sort on the result. Show that the worst-case running time is still $O(n^2)$, but try to compare the modified algorithm with the traditional insertion sort to see if it works better in practice.

Bucket sort

Exercise: Argue why the inner loop in

```
result_keys, result_values = [], []
for key in range(m):
 for val in buckets[key]:
  result_keys.append(key) #This line
  result_values.append(val) #And this line
```

only executes n times.

Exercise: Argue why the bucket sort actually sorts the input.

If you want to see a more efficient, and more traditional, bucket sort, then try this exercise.