CSIT113
Problem Solving

Week 8

Searching

- Assume that we have to find whether a particular name is in a list of names.
- What is our best strategy?
- This depends on the precise nature of the list.
- OK – the list is in no special order.
- Now what is our best strategy.

Searching an unordered list.

- In this case, the “best” strategy is actually not especially good.
- Look at each item, in turn, until we find the one we are looking for or we run out of names.
- This is known as a linear search and, on average, we will look at around half of the names in the list (assuming it is there at all).
- In the worst case, the name is not on the list, it involves looking at all of the entries.

Improving on linear search.

- What can we do to the list to make it easier (more efficient) to search?
- Think about lists of words you are familiar with.
  - Phone books;
  - Dictionaries;
  - Indexes.
- What property do all these lists share?
- They are in alphabetical order!
- Why is this done?

Searching ordered lists.

- If a list is in order we can search it more efficiently.
- We do need to know how big the list is, though.
- Here is our new strategy: **Binary Search**
  1. Look at the word in the middle of the list.
  2. If this word is the one we are looking for, stop.
  3. If this word is after the one we are looking for, replace the list with the first half of the list.
  4. If this word is before the one we are looking for, replace the list with the second half of the list.
  5. Go back to step 1. with the shorter list.

Comparison.

- Is this a better strategy?
- Let's consider how much of the list we eliminate at each step in the strategy.
- Linear search:
  - 1 word less at each step.
  - Reduce and conquer, step size of 1.
- Binary search:
  - Halve the number at each step.
  - Reduce and conquer, step size $n/2$.

An example.

- A recent estimate stated that the number of words in the English language is 1,025,109.8
  - I’m not sure how you get 4/5 of a word either!
- If I start to look for a random word in this list using a linear search and can check one word every second, without error, I can expect to be done in 1,025,109.8/2 seconds.
- That is around six days!
  - If I don’t take any breaks to eat, drink, sleep etc.
  - 12 days in the worst case.
- What about binary search?

- We can see what will happen with binary search if we build a table.

Number of checks	Size of list remaining	Total time
0	1,025,109.8	0 sec
1	512555	1 sec
2	256277	2 sec
3	128138	3 sec
4	64069	4 sec
5	32034	5 sec
6	16017	6 sec

- How long will it take to get to a list of size 1?
- 20 seconds!

- So, that is 6 days or 20 seconds.
- I know which looks better to me!
- Hang on, though!
- I may have cheated.
- Can you see how?
- How did the list get sorted?
- How long did that take?

Sorting a list, a good idea?

- As we will soon see, it takes a lot of work to sort a list.
- Much more than it takes to search it!
- Even with linear search!
- This means that we would be crazy to sort the list before searching it.
- Except…
- what if we want to search it many times?
- We only have to sort it once.

- So, provided we are going to search the list a lot, it makes sense to sort it.
  - Dictionaries.
  - Telephone books.
- Otherwise, don’t bother.
- So… we have decided the list is worth sorting.
- How do we do it?
- We have lots of choices…

All sorts of sorts.

- We will start by looking at three simple sorting strategies:
  - Selection sort.
  - Insertion sort.
  - Bubble sort.
- Each strategy works in a slightly different way but includes two basic operations:
  - Compare two items in a list.
  - Swap two items in a list.
- What changes is the way these two operations are used.

What to sort.

- For the examples of sorting in this lecture I will use a list of positive integers instead of words.
- Why?
  - They are easier to type.
  - They take up less room.
  - It is easier to check if they are in order.
- The same sorting strategies will work on words too.
- In fact they will work on any list as long as we have well-defined compare and swap operations.

Selection Sort.

- Selection sort uses the following strategy:
  1. Start with the whole list.
  2. Find the smallest element in the list.
  3. Swap the first element with the smallest.
  4. Shorten the list by ignoring its first element (because it is in the right place).
  5. If the list has only one element, stop.
  6. Otherwise, go to step 1. with the shorter list.
- We can see how this works with an example.

Selection sort example.

- Starting list:

14	20	7	8	5	15	18	17	6	13

- Where does the list start?

14	20	7	8	5	15	18	17	6	13

- Where is the smallest number?

14	20	7	8	5	15	18	17	6	13

- Swap these numbers.

5	20	7	8	14	15	18	17	6	13

- Shorten the list by one.

5	20	7	8	14	15	18	17	6	13

Selection sort example.

- Starting list:

5	20	7	8	14	15	18	17	6	13

- Where does the list start?

5	20	7	8	14	15	18	17	6	13

- Where is the smallest number?

5	20	7	8	14	15	18	17	6	13

- Swap these numbers.

5	6	7	8	14	15	18	17	20	13

- Shorten the list by one.

5	6	7	8	14	15	18	17	20	13

Selection sort example.

- Starting list:

5	6	7	8	14	15	18	17	20	13

- Where does the list start?

5	6	7	8	14	15	18	17	20	13

- Where is the smallest number?

5	6	7	8	14	15	18	17	20	13

- It’s the same number. No need to swap.

5	6	7	8	14	15	18	17	20	13

- Shorten the list by one.

5	6	7	8	14	15	18	17	20	13

Selection sort example.

- Starting list:

5	6	7	8	14	15	18	17	20	13

- Where does the list start?

5	6	7	8	14	15	18	17	20	13

- Where is the smallest number?

5	6	7	8	14	15	18	17	20	13

- It’s the same number. No need to swap.

5	6	7	8	14	15	18	17	20	13

- Shorten the list by one.

5	6	7	8	14	15	18	17	20	13

Selection sort example.

- Let's speed this up

5	6	7	8	14	15	18	17	20	13

- Each cycle

5	6	7	8	13	15	18	17	20	14
5	6	7	8	13	14	18	17	20	15
5	6	7	8	13	14	15	17	20	18
5	6	7	8	13	14	15	17	20	18
5	6	7	8	13	14	15	17	18	20
5	6	7	8	13	14	15	17	18	20

- And we are done!

How hard was that?

- To sort a list of 10 numbers we completed 9 cycles.
- But I cheated again!
- How did I find the smallest number each time?
- What I really need to do involves a bit (a lot) more work.

1. Assume the first element in the list is the smallest.
2. Note its value and location.
3. Compare the smallest value with each element in the list.
4. If it is smaller remember its value and location instead.

- Let’s see that in action.
- Starting position:

5	6	7	8	13	14	18	17	20	15

- Smallest = 18
- Look at next element, is it smaller?

5	6	7	8	13	14	18	17	20	15

- Yes, new Smallest = 17

5	6	7	8	13	14	18	17	20	15

- Still 17

5	6	7	8	13	14	18	17	20	15

- Smallest =15, and we are done.
- Now we can swap 18 and 15.

How much work?

- So, to find the smallest element in each cycle we perform a comparison for each item left in the unsorted list.
- To sort a list of 10 items that is 9+8+…+2+1 = 45 comparisons.
- And up to 9 swaps.
- For a list of N numbers we will carry out:
- $N \times (N - 1) / 2$ comparisons and $N - 1$ swaps.
- That is roughly $N^2 / 2$ operations.
- Perhaps we can do better.

Insertion Sort.

- Insertion sort uses the following strategy:
    1. Start with the second element in the list.
    2. Insert it in the right place in the preceding list.
    3. Repeat with the next unsorted element.
    4. Keep going until we have placed the last element in the list.
- We can see how this works with an example.

Insertion sort example.

- Starting list:

14	20	7	8	5	15	18	17	6	13

- Start at the second element.

14	20	7	8	5	15	18	17	6	13

- It's in the right place

14	20	7	8	5	15	18	17	6	13

- We have finished the step.

14	20	7	8	5	15	18	17	6	13

- Repeat.

Insertion sort example.

- Starting list:

14	20	7	8	5	15	18	17	6	13

- Start at the next element.

14	20	7	8	5	15	18	17	6	13

- Put it in the right place.

7	14	20	8	5	15	18	17	6	13

- We have finished the step.

7	14	20	8	5	15	18	17	6	13

- Repeat.

Insertion sort example.

- Starting list:

7	14	20	8	5	15	18	17	6	13

- Start at the next element.

7	14	20	8	5	15	18	17	6	13

- Put it in the right place.

7	8	14	20	5	15	18	17	6	13

- We have finished the step.

7	8	14	20	5	15	18	17	6	13

- Repeat.

Insertion sort example.

- Starting list:

7	8	14	20	5	15	18	17	6	13

- Start at the next element.

7	8	14	20	5	15	18	17	6	13

- Put it in the right place.

5	7	8	14	20	15	18	17	6	13

- We have finished the step.

5	7	8	14	20	15	18	17	6	13

- Repeat.

Insertion sort example.

- Again, lets speed it up.

5	7	8	14	20	15	18	17	6	13

- Each Cycle.

5	7	8	14	15	20	18	17	6	13
5	7	8	14	15	18	20	17	6	13
5	7	8	14	15	17	18	20	6	13
5	6	7	8	14	15	17	18	20	13
5	6	7	8	13	14	15	17	18	20

- And we are done

How hard was that?

- Again, to sort a list of 10 numbers we completed 9 cycles.
- But I cheated again!
- This time, how did I get the number into the right place?
- Again, I need to do more work.

1. Find where the new element should be in the list so far.
2. Move every element above this up by one place.
3. Move the new element into the hole.

- Let’s see that in action.
- Starting position:

7	14	20	8	5	15	18	17	6	13

- We need to insert the value 8, where does it go?

7	14	20	8	5	15	18	17	6	13

- Here!
- Move the elements up by 1.

7	14		20	5	15	18	17	6	13
7		14	20	5	15	18	17	6	13

- Insert the 8

7	8	14	20	5	15	18	17	6	13

Another way to do it.

- We can combine the two steps:
  - Find the right place;
  - Put the element there.
- To do this we simply swap the new element down until it is in the right place.
- This, however, replaces moves with swaps which are slower.

How much work this time?

- So, to find the smallest element in each cycle we perform a comparison for each item in the sorted part of the list that is greater then the new item.
- This will involve, in the worst case, looking at each element.
- To sort a list of 10 items that is 1 + 2 + 3 + … + 8 + 9 = 45 comparisons.
- The same number of moves.
- For a list of *N* numbers we will carry out:
- $N \times (N - 1) / 2$ comparisons and up to $N \times (N - 1) / 2$ swaps.
- That is roughly N^2 operations.
- This does not look a lot better.

Bubble Sort.

- Bubble sort uses the following strategy

1. Compare the first element with the next one.
2. If they are in the wrong order swap them.
3. Continue comparing until the end of the list.
4. Shorten the list by one – the last element is now in the right place.
5. Repeat from step 1.

- If you don’t do any swaps on any cycle you stop.

Bubble sort example.

- Starting list:

14	20	7	8	5	15	18	17	6	13

- Compare elements 1 and 2.

14	20	7	8	5	15	18	17	6	13

- No swap needed. Now compare elements 2 and 3.

14	20	7	8	5	15	18	17	6	13

- Swap them. Compare elements 3 and 4.

14	7	20	8	5	15	18	17	6	13

- Repeat.

Bubble sort example.

- Keep going, swapping as needed.

14	7	20	8	5	15	18	17	6	13
14	7	8	20	5	15	18	17	6	13
14	7	8	5	20	15	18	17	6	13
14	7	8	5	15	20	18	17	6	13
14	7	8	5	15	18	20	17	6	13
14	7	8	5	15	18	17	20	6	13
14	7	8	5	15	18	17	6	20	13
14	7	8	5	15	18	17	6	13	20

- That is the first pass complete.

- Let’s do it again.

14	7	8	5	15	18	17	6	13	20
7	14	8	5	15	18	17	6	13	20
7	8	14	5	15	18	17	6	13	20
7	8	5	14	15	18	17	6	13	20
7	8	5	14	15	18	17	6	13	20
7	8	5	14	15	18	17	6	13	20
7	8	5	14	15	17	18	6	13	20
7	8	5	14	15	17	6	18	13	20
7	8	5	14	15	17	6	13	18	20

- That’s two passes.

- Let’s just look at the end of each pass

7	8	5	14	15	17	6	13	18	20
7	5	8	14	15	6	13	17	18	20
5	7	8	14	6	13	15	17	18	20
5	7	8	6	13	14	15	17	18	20
5	7	6	8	13	14	15	17	18	20
5	6	7	8	13	14	15	17	18	20
5	6	7	8	13	14	15	17	18	20

- We didn’t swap anything that time. We are finished.

5	6	7	8	13	14	15	17	18	20

How hard this time?

- Again, to sort a list of 10 numbers we complete at most 9 cycles.
- For each cycle we performed one less comparison.
- We also perform up to the same number of swaps.
- That is 9 + 8 + 7 + …. + 2 + 1 = 45 comparisons.
  - Actually, it was less than that because we stopped early.
- And some number of swaps, again at most 45.
- So, that is up to $N \times (N - 1) / 2$ comparisons and $N \times (N - 1) / 2$ swaps.
- Again, about N^2 operations.

What is so special about N^2 operations?

- Notice that the number of operations in all of these sorts involves around N^2 operations.
- Ok, selection sort uses half as many in the worst case.
- Does this mean all sorts take around N^2 operations?
- No.
- Notice that all three sorts we have looked at use reduce and conquer.
- We can do better.
- We can also do a lot worse!
- Let us look at one last sort.

QuickSort

- The quicksort strategy is a bit more difficult to understand but let us first try and explain it in English.
- Split the list into two parts.
- The first part contains all the elements smaller than the original first element.
- The second part contains all the elements greater than or equal to the original first element.
- Repeat on each part.
- Keep splitting each sub list into two parts until we have a sorted list.

Quicksort example.

- Starting list:

14	20	7	8	5	15	18	17	6	13

- Partition the list. Note that the 14 is in the right place.

6	13	7	8	5	14	18	17	15	20

- Partition the first half.

5	6	7	8	13	14	18	17	15	20

- Partition again. Only one element so it is in the right place.

5	6	7	8	13	14	18	17	15	20

- Partition the next sub list. Nothing moved so 7 is in the right place too.

5	6	7	8	13	14	18	17	15	20

5	6	7	8	13	14	18	17	15	20

- Partition the next sub list. Nothing moved so 8 is in the right place too.

5	6	7	8	13	14	18	17	15	20

- Partition again. Only one element so it is in the right place.

5	6	7	8	13	14	18	17	15	20

- Partition the second half of the list.

5	6	7	8	13	14	15	17	18	20

- Partition the sub list. Nothing moved so 15 is in the right place.

5	6	7	8	13	14	15	17	18	20

- Partition again. Only one element so it is in the right place.

5	6	7	8	13	14	15	17	18	20

- Partition the last part. Only one element so it is in the right place.

5	6	7	8	13	14	15	17	18	20

How much work this time?

- To perform each partition we need to look at each element in the sub list.
- Each time the sub lists split into two parts. Ideally, into halves.
- We can show that quicksort uses around $N \times \log(N)$ operations.
- This is considerably better than N^2 operations!
- So, should we always use quicksort?

Which sort is best?

- The best sort to use depends on a number of factors.
- How many items are we sorting?
  - If the number of items is small, quicksort is overkill.
- How disordered is the list?
  - If the list is nearly in order insertion sort or bubble sort work really well.
- If there are only a small number of items out of order:
  - Use insertion sort.
- If the items are nearly in order:
  - Use bubble sort.