Mergesort (and bubble sort)

Dr Tom Ridge

2018 Q1

Introduction

- What is the default algorithm that Perl, Java, etc use for sorting? Merge sort (for Java, this was recently changed in Java 7 to TimSort, which is a variant of merge sort)
- The aim of the next few slides is to introduce the merge sort algorithm.
- "Merge sort" is commonly written as a single word: "mergesort"
- Mergesort is a very cool algorithm. Amazingly it was invented in 1945 (before they even had real computers!)
- https://en.wikipedia.org/wiki/Merge_sort

Plan for this lecture

- First look at some functions that are used by merge sort
- Then look at merge sort itself

Auxiliary functions needed for merge sort

- `take` and `drop` - `take(n,l)` returns first `n` elements of the list `l`; `drop` drops the first `n`
- `merge` - take two sorted lists and return a sorted list

Take and drop

Take

`take(int n, List l)` returns the first `n` elements of `l`

```
// return first n elts
List<Integer> take(int n, List<Integer> l) {
  List<Integer> to_return = nil();
  while(true) {
    if(length(to_return) == n) return to_return;
    to_return = append1(to_return,hd(l));
    l = tl(l);
  }
}
```

How does `take` execute?

Drop

`drop(int n, List l)` drops the first `n` elements of `l`

```
List<Integer> drop(int n, List<Integer> l) {
  List<Integer> to_drop = nil();
  while(true) {
    if(length(to_drop) == n) return l;
    to_drop = append1(to_drop,hd(l));
    l = tl(l);
  }
}
```

Merge 2 ordered lists

Write a function `List merge(List l1, List l2)` that takes two sorted lists of integers and returns a single sorted list. For example `merge([1,2,4],[3,5])` should give the list `[1,2,3,4,5]`.

How does merge execute?

Merge 2 ordered lists

```
List<Integer> merge(List<Integer> l1, List<Integer> l2) {
  List<Integer> to_return = nil();
  while(true) {
    if(l1.isEmpty()) return append(to_return,l2);
    if(l2.isEmpty()) return append(to_return,l1);
    int i1 = hd(l1);
    int i2 = hd(l2);
    if(i1 < i2) {
      to_return = append1(to_return,i1);
      l1 = tl(l1);
    } else {
      to_return = append1(to_return,i2);
      l2 = tl(l2);
} } }

l1 = cons(1,cons(3,cons(5,nil())));
l2 = cons(2,cons(4,cons(6,cons(8,nil()))));
merge(l1,l2);
```

Merge sort

Example: 7 4 3 1 2 6 9 5
Idea: break the problem down into pieces; when the pieces are small enough, solve them outright; combine the solutions of the pieces to get a solution to the whole
This is a general approach to algorithms, called "Divide and conquer"
https://en.wikipedia.org/wiki/Divide_and_conquer_algorithms
See Wikipedia page: http://en.wikipedia.org/wiki/Merge_sort
I'll draw on the board/visualizer how merge sort might work on this input

Merge sort

```
List<Integer> mergesort(List<Integer> l) {
  if(length(l)==0 || length(l)==1) return l; // already s
  List<Integer> l1 = take(length(l)/2,l);
  List<Integer> l2 = drop(length(l)/2,l);
  l1 = mergesort(l1);
  l2 = mergesort(l2);
  return merge(l1,l2);
}
```

The full code for mergesort is available from the module webpage. N.B. this is not an efficient implementation of merge sort (see module CO3002 for efficient code). But it does allow us to understand the behaviour of merge sort.

Merge sort function calls

- sort some numbers: 7 4 3 1 2 6 9 5

Worst case performance

- Why prefer mergesort to, say, insertion sort?
- It is because of the performance: mergesort (properly implemented) is "quicker" than insertion sort.
- In order to understand performance, you need to understand big-Oh notation, which is covered in CO3002 (in 2 years time).
- However, looking at the wikipedia page, we can see that mergesort has a worst case performance of `O(n log n)` which should be read as `O(n * (log n))`
- Insertion sort has a worst case performance of `O(n^2)` i.e. `O(n*n)`.
- Here the logarithm `log n` of a number can be interpreted eg as: `log 4 = 2; log 8 = 3` i.e. it is the number `x` such that `2^x = n`
- Let `f(n) = n * log n` and `g(n) = n * n`. Then `f(n)` is "smaller than" `g(n)`
  - and so the worst case performance of mergesort is "smaller than" the worst case performance of insertion sort

Single example, comparison of insertion sort and mergesort (only for those who are really interested)

- Count the number of integer comparisons for mergesort and insertion sort on list `[2,4,8,7]`
- Mergesort: 4
- Insertion sort: 6
- So mergesort is better in this case, at least if we consider the number of integer comparisons

The implementation above is inefficient

- I emphasize that the implementations given above are **not** efficient.
- For example, repeatedly traversing a list to find its length is too inefficient.
- The implementations above are intended to give a flavour for what happens, but real implementations must overcome engineering issues that would complicate the above presentation.

Bubble sort

- Bubble sort is included in the syllabus because it is well-known
- Bubble sort is a terrible sorting algorithm
- See Wikipedia "Bubble sort" article: "the archetypical perversely awful algorithm" https://en.wikipedia.org/wiki/Bubble_sort

Bubble sort

Bubble sort in Java

Warm up... thinking about array indexing

```
int[] a = { 3,1,10,11,9,4 };

for(int i = 0; i < a.length; i++) {
  print(a[i]);
}
```

```
for(int i = 0; i < a.length -1; i++) {
  print(a[i]+" "+a[i+1]);
}
```

Bubble sort in Java, inner loop

```
for(int i = 0; i < a.length -1; i++) {
  if(a[i] > a[i+1]) {
    // swap a[i] and a[i+1]
    int tmp = a[i];
    a[i] = a[i+1];
    a[i+1] = tmp;
  }
}
```

- What happens to the biggest element in the array? it ends up in the right place

Bubble sort in Java

- The biggest element ended up in the right place.
- Suppose we do this loop again, what happens to the second biggest element in the array? (it ends up in the right place)
- If there are n elements in the array, and we do the loop n times, then ... the biggest n elements are in the right place (i.e. all of them); in fact, we really only have to do the loop n-1 times (since if n-1 elements are in the right position, then n elements are in the right position)

Bubble sort in Java

```
for(int j=0; j < a.length; j++) {
  for(int i = 0; i < a.length -1; i++) {
    if(a[i] > a[i+1]) {
      // swap these entries
      int tmp = a[i];
      a[i] = a[i+1];
      a[i+1] = tmp;
    }
  }
}
```

- The above algorithm is known as "bubble sort"