Lecture 12: Sorting Review and Other List Algorithms

Review: QuickSort

- let p=hd(l)
- split L into to L1= elements in L <=p, L2= elements in L > p
- [10,2,7,12,28,22] —> p=10 [2,7] [12, 28, 22]
- sort the lists recursively — let them return S1 & S2
- return S1@ (p::S2)
  - in this ex R1=[2,7], R2= [12,22,28]
  - [2,7] @ [10,12,22,28] = [2,7,10,12,22,28]

Quicksort

```
fun qsort([]) = []
    | qsort(x::xs) =
        let
            fun comp opr x y = opr(y,x):bool
        in
            qsort(x::xs) = qsort(List.filter (comp op<= x) xs) @ (x::qsort(List.filter (comp op> x) xs));
       end;
```

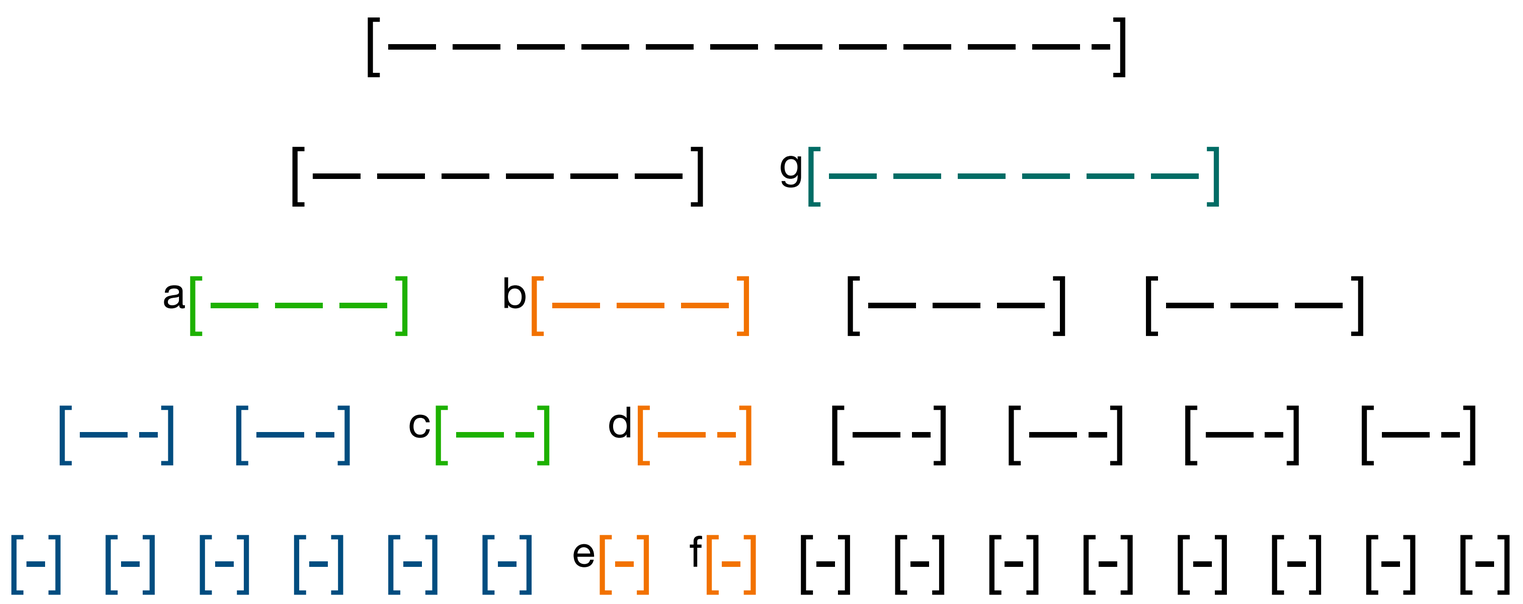
Complexity of Quicksort

- split & append each can take O(n)
- what is the depth? if depth O(d) complexity is O(n d)
- depth depends on how many elements in L1 & L2? Do they divide evenly?
- worst case is when list is already sorted —- we get a division of 0,n-1 each time

Reiew: Mergesort Space Complexity

- depth is log n
- After split we get two new arrays of size n/2.
  - but we can forget the original list of size n
- After merge we get a new array of size n
  - but we can forget the two arrays we merged
- what is space complexity?

MERGE SORT SPACE USAGE



total storage n+n/2+n/4 .. 1 = 2n = O(n)

Quick Sort Space Complexity

?????

Discuss in Interactive session

More List Examples

Define a function first_`list: ('a * 'b) list -> 'a list` which takes in input a list of tuples and gives back the list consisting of the first elements only of each tuple,

Examples:

```
first_list [] = []
first_list [(1,2),(1,3)] = [1,1]
first_list [(1,"a"),(2,"b"),(3,"c")] = [1,2,3]
first_list [([],"a"),([1],"b"),([1,2],"c")] = [[],[1],[1,2]]
```

First List

```
fun first_list [] = []
      | first_list((x,y)::l) = x::first_list l;
```

How about using map?

Zip/Unzip lists

Define a function zip_list: takes two equal length lists and creates a list tuples (a,b) where a is with ite fro list 1 and b is with its from list 2

Examples:

```
zip([1,2,3], [“a”,”b”,”c”]) =[(1,”a"),(2,"b"),(3,"c")]
```

ZIP List

```
fun zip_list ([],[]) = []
     | zip_list((x::xs, y::ys) = (x,y)::zip_list (xs,ys);
```

what about reverse operation unzip? Should return pair of lists.

Flatten a list

Task

Write a function to flatten the nesting in an arbitrary list of values.

Your program should work on the equivalent of this list:

```
[[1], 2, [[3, 4], 5], [[[]]], [[[6]]], 7, 8, []]
```

Where the correct result would be the list:

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

Datatype (recursive new type)

- Lists are one example of the notion of a *recursive datatype*. ML provides a general mechanism, the `datatype` declaration, for introducing recursive types. Earlier we introduced `type` declarations as an abbreviation mechanism. Types are given names as documentation and as a convenience to the programmer, but doing so is semantically inconsequential --- one could replace all uses of the type name by its definition and not effect the behavior of the program. In contrast the `datatype` declaration provides a means of introducing a *new* type that is distinct from all other types and that does not merely stand for some other type. It is the means by which the ML type system may be extended by the programmer.

```
datatype 'a llist = LList of 'a llist list| Elem of 'a;
```

Flatten logic

```
datatype 'a llist = LList of 'a llist list| Elem of 'a;

fun flatten (Elem x) = [x]
  | flatten (LList []) = []
  | flatten (LList (LL::LLs)) = flatten LL @ flatten (LList LLs)
```

Longest Common Subsequence

- L1=[3,2,11,8,23,16,9] L2[7,2,14,8,11,16,2,9,23]
- Find longest common subsequence in L1 & L2
  - here [2,8,16,9], [2,11,16,9], [2,11,23] are example of common subsequence
- recursive logic
  - LCS(a::as, b::bs)
    - if a=b then a::LCS(as,bs)
    - else longest of LCS(a::as,bs) LCS(as, b::bs)
  - base case LCS([],b) or LCS[a,[]) is []

Complexity of Recursive LCS

- lists of size men then recursive depth can be m+n-1
- At every level two recursive calls so options double
- so t most 2^(m+n-1)