Sorting

Sorting is an essential activity in Data Processing
e.g. In Binary Search the array is sorted

The methods for Sorting depend on

Data Structures -- arrays, lists, files

The data structures may be stored <u>internally</u> or <u>externally</u>.

Internal:

Sorting is done in memory; this gives fast access to the items e.g. arrays

Consideration is also given as to whether the sorting is done in-place or not. An in-place sort of a sequence of n items uses a constant amount of extra space. The following algorithm for sorting is not in-place

Algorithm:
Copy smallest item to 1st position in another array,
copy next smallest to 2nd position
etc.

We will be concerned with in-place sorting of arrays.

<u>External</u>:

The sequence to be sorted is stored externally on files e.g. disk or tape.

Performance -- $O(n^2)$ .v. $O(n * (\log n))$

Algoritms for sorting can be classified according to whether they are

$O(n^2)$ -- Elementary / Simple sorts or

$O(n * \log n)$ -- Advanced / Sophisticated sorts

Sort algorithms can be grouped according to strategies;

1. **Insertion: e.g.** Insert sort -- $O(n^2)$
   Merge sort -- $O(n*(\log n))$

2. **Selection: e.g.** Select sort -- $O(n^2)$
   Heapsort / Treesort -- $O(n*(\log n))$

3. **Exchange: e.g.** Bubble / Exchange -- $O(n^2)$
   Quicksort -- $O(n*(\log n))$

A sort like Shell Sort is difficult to classify even though it may be considered an Insertion sort but it is neither $O(n^2)$ nor $O(n*(\log n))$.

Also, Select sort may be thought of as a worst case of Quicksort.

Other considerations

- **the data may be stored in sophisticated data structures e.g. Binary Trees or Priority Queues.**
- **the sort should be stable, i.e. the relative order of equal items is unchanged.**
- **The sort should be Natural, i.e. it sorts almost sorted sequences best.**

Order of Execution Time -- the O(n) notation

Let f(n) and g(n) be two functions. We say that f(n) is (no more than) Order g(n), written O(g(n)), if there exists
c > 0 s.t. for all (except maybe a finite number) naturals n

$$f(n) \le c_* g(n)$$

We say f(n) is proportional to g(n) iff f(n) is O(g(n)) and g(n) is O(f(n))
i.e. the are the same order.

e.g. 1

f(n) = n+5; g(n) = n

then O(fn)) = O(g(n))

To show f(n) is O(g(n)), choose c = 5

$f(n) = n+5 \le 5*n = c*g(n), \quad n>1$

To show g(n) is O(f(n)), choose c=1

$g(n) = n \le n+5 = c*f(n)$

<u>e.g. 2</u>

Is 3^n $O(2^n)$? No!

Suppose ∃ (“there exists”) n_0, k s.t. for all $n \ge n_0$, $3^n \le c*2^n$

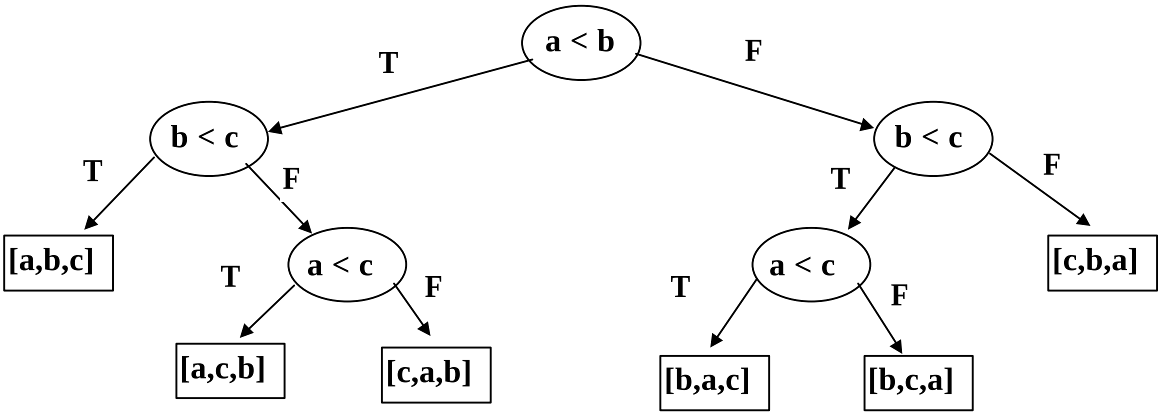
Then $c \ge \left(\frac{3}{2}\right)^n$ **for any** $n \ge n_0$

This is impossible as $\left(\frac{3}{2}\right)^n$ **gets arbitrarily large.**

Theoretical Optimum for Sorting.

This assumes we are using <u>binary comparison</u>. The optimal number of comparisons is O(n_*Log n).

Assume the sequence has n distinct elements.
e.g. n = 3
s has values a,b,c. Let use the notation [a,b,c] for the sequence s.



Height of Binary Tree with k leaves = $\lceil \log k \rceil$ "ceiling(log k)"

tf.

#Comparisons $\geq$ log n!

but log n! = log n + log (n-1) +... + log 2

tf.

log n! is O(n*log n)

Sequence

From mathematics, a sequence of n items of type T is a function from Nat_n to T.

e.g. $\quad x : Nat_n \rightarrow Real$

$$k \mapsto x(k)$$

where $Nat_n = \{1,2,...,n\}$

We can use the notation

$[x(1),..., x(n)]$ **or** $[x_1,.., x_n]$

to denote the sequence x.

Permutation of a Sequence

Let X, Y be 2 sequences, then define

Perm(X,Y) "Y is a Permutation of X"

$\equiv \exists$ **a bijective function p on** Nat_n **s.t.** $Y = [x(p(1)), x(p(2)),..., x(p(n))]$

e.g. **Let X and Y be sequence of characters where X = [a,a,b] and Y = [a,b,a]**

tf. **Perm([a,a,b], [a,b,a])**

Let p $= \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$

$$\begin{aligned} [a,b,a] &= [x(1), x(3), x(2)] \\ &= [x(p(1)), x(p(2)), x(p(3))] \end{aligned}$$

Another Defn of Perm(L,M)

Def^n: Bag or Suite

A bag is a multi-set, i.e. a 'set' with repeated elements. More precisely, if B is a bag of elements of type T then

$B : \quad T \rightarrow Nat$

$$t \mapsto B(t),$$

where B(t) is the number of occurrences of t in B.

e.g. 1 Let T be the type CHARACTER and

$$B = \{a, a, c, a, c\}$$

then B(a) = 3, B(b) = 0, B(c) = 2 and B(x) = 0, for other x : T

e.g. 2 Let T be the digits 0 .. 9 then

and B = { 1, 5, 2, 3, 9, 9, 0, 2, 5 }

B(0) = 1, B(1) = 1, B(2) = 2, B(3) = 1, B(4) = 0,

B(5) = 2, B(6) = 0, B(7) = 0, B(8) = 0, B(9) = 2

Sequence and Bags / Suites

If s is a sequence of n items of type T

i.e. $s : Nat_n \rightarrow T$

then **Bag(s) is the bag of items in s.**

$Bag(s) : T \rightarrow Nat_n$

where $Bag(s)(t) = \#\{k \in Nat_n \mid s(k) = t\}$

e.g. $s: Nat_n \rightarrow CHARACTER$

s.t. **s = [a,b,a]**

then $Bag(s) : CHARACTER \rightarrow Nat_n$

i.e. Bag(s) = {a,b,a}

We can define Bag(s) in terms of a function count(t,s) which returns the numbers of occurrences of t in s.

More precisely, let B = Bag(s), then $B : T \rightarrow Nat_n$
where B(t) = count(t,s) i.e. Bag(s)(t) = count(t,s)

We define count(t,s) recursively,

Notation:

Let [ ] be the empty sequence, and
let t r s be 'item t prepended to sequence s'

count : T x SEQ[T] f Nat
-- SEQ[T] " sequence of items from T"

count(t, [ ]) = 0

count(t, t r s) = 1 + count(t,s)

count(t, u r s) = count(t,s), if t ≠ u

Note:

We can regard a <u>set</u> S of elements of T as

$$S : T \to \text{BOOLEAN}$$

where $t \in S \equiv S(t)$

Equality of Bags

B1 = B2 as bags

iff B1 and B2 have the same elements with the same occurrences.

$B1 = B2 \equiv (\text{All } x{:}T \mid : B1(x) = B2(x))$

i.e. B1 = B2 as functions.

Defn #2 Perm(L,M)

Let L and M be sequences of type T. The sequences L and M may have repeated elements.

Let Bag(L) be the bag corresponding to L

We define

$$\begin{aligned}\text{Perm}(L,M) &\equiv (\text{All } x{:}T \mid : \text{count}(x,L) = \text{count}(x,M)) \\ &\equiv \text{Bag}(L) = \text{Bag}(M)\end{aligned}$$

Def^{n} of Sorting a Sequence

$$M = \text{Sort}(L) \equiv \text{Perm}(L, M) \;\&\; \text{Ordered}(M)$$

where as before,

$$\text{Ordered}(M) = (\text{All } k \mid 1 \le k < n : M(k) \le M(k+1))$$

"Extremely Slow Sort"

To sort L, generate all Perms of L and check which one is Ordered.