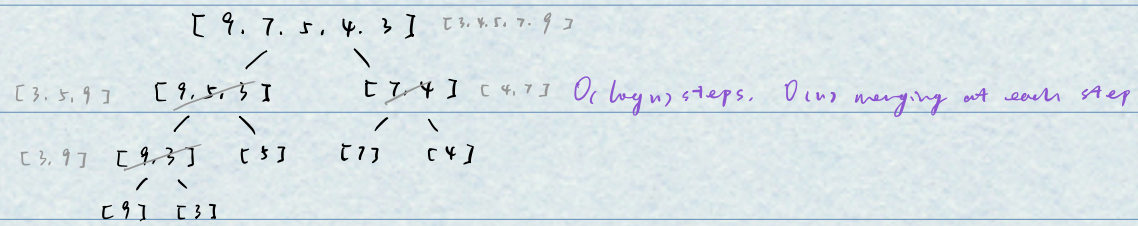
Recall: Int.compare : int * int → order

insertion sort for lists $O(n^2)$, n = length of list



$W(n) = c \cdot n + 2 \cdot W(\frac{n}{2}) \Rightarrow W(n) = O(n \log n)$

Merge Sort (List)

```
(* msort : int list -> int list
   REQUIRES: true
   ENSURES: msort(L) returns a sorted permutation of L
*)
fun msort ([] : int list) : int list = []
  | msort ([x]) = [x]
  | msort (L) = let
                  val (A, B) : int list * int list = split L
                in
                  merge (msort A, msort B)
                end

(* split : int list * int list
   REQ : true
   ENS : split L ≅ (A, B) s.t. L ≅ permutation of A @ B & |length A - length B| ≤ 1
*)
fun split ([] : int list) : int list * int list = ([], [])
  | split ([x]) = ([x], [])
  | split (x::y::L) = let
                        val (A, B) : int list * int list = split L
                      in
                        (x::A, y::B)
                      end
```

=> $O(n)$

$W(n)$

$W(0) = C_0$

$W(1) = C_1$

$W(n) = C_2 + W(n-2)$

$= \underbrace{C_2 + C_2 + C_2 + \cdots}_{\frac{n}{2}} + \begin{Bmatrix} C_0 \\ C_1 \end{Bmatrix} = \frac{n}{2} \cdot C_2 + \begin{Bmatrix} C_0 \\ C_1 \end{Bmatrix}$ is in $O(n)$

```
(* merge : int list * int list -> int list
   REQ : A, B are sorted
   ENS : merge(A, B) ≅ L, L is a sorted permutation of A @ B
*)
fun merge ([] : int list, B : int list) : int list = B
  | merge (A, []) = A
  | merge (x::A, y::B) = (case Int.compare (x, y) of   (* make code more general (we can use any type t.compare) instead of x < y. *)
                            LESS => x::merge(A, y::B)
                          | EQUAL => x::y::merge(A, B)
                          | GREATER => y::merge(x::A, B))
```

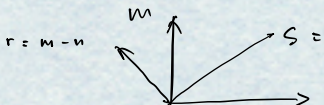
=> $O(n)$ throw away 1/2 element(s) every iteration

$W_{merge}(n, m)$

$W(0, m) = C_0$ for all $m \geq 0$

$W(n, 0) = C_1$ for all $n > 0$

$W(n, m) = \begin{cases} C_2 + W(n-1, m) & \text{if LESS} \\ C_2 + W(n-1, m-1) & \text{if EQUAL} \\ C_2 + W(n, m-1) & \text{if GREATER} \end{cases}$



$W_{merge}(s)$

$s \geq 0$ but one of m or n is 0 --- $W(s) = k_0$

$s > 0$ & $m > 0$ & $n > 0$ --- Worst case $W(s) \leq k_1 + W(s-1)$

$\Rightarrow W(s)$ is $O(s) = O(m+n)$

Finally, $W_{msort}(n)$

$W_{msort}(0) = C_0$

$W_{msort}(1) = C_1$

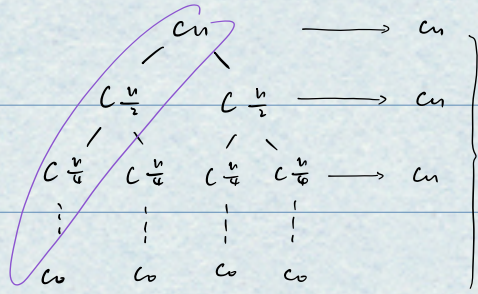
$W_{msort}(n) = C_2 + \underbrace{W_{split}(n)}_{O(n)} + 2W_{msort}(\frac{n}{2}) + \underbrace{W_{merge}(n)}_{O(n)}$

$\leq C_2 + C_3 n + 2W_{msort}(\frac{n}{2})$

$\leq C_4 n + 2W_{msort}(\frac{n}{2})$

$S_{msort}(n) = C_5 + \max \text{ of } \{ S_{msort}(\frac{n}{2}), S_{msort}(\frac{n}{2}) \}$

$S_{msort}(n) = C_5 + S_{msort}(\frac{n}{2})$



# of levels is $\log_2 n$

$\Rightarrow W_{msort}(n) = O(n \log n)$

span

$C_5 + C\frac{n}{2} + \cdots + C_0$

$= C_5(1 + \frac{1}{2} + \frac{1}{4} + \cdots)$ $\Rightarrow$ span is $O(n)$

$\leq C_5 n$

Merge Sort (Tree)

```
datatype tree = Empty | Node of tree * int * tree
```

T is sorted if either T is empty or T is Node(l, x, r) with l sorted & every element in l is LESS/EQUAL to x & every element in r is GREATER/EQUAL to x


```
(* Ins : int * tree -> tree
   REQ : T is sorted
   ENS : Ins(x, T) is a sorted tree containing exactly x & all ele in T
*)
fun Ins (x : int, Empty : tree) : tree = Node(Empty, x, Empty)
  | Ins (x, Node(l, y, r)) = (case Int.compare(x, y) of
                                GREATER => Node(l, y, Ins(x, r))
                              | _       => Node(Ins(x, l), y, r))
```

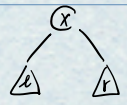
Work : $O(d)$ sorting by inserting : up to $O(n^2)$

Span : $O(d)$ balanced : $O(n \log n)$

Recall list insertion : $O(n^2)$ [n = # elements] tree insert : work $O(d)$ [d = # depth]

We will do better today : $O((\log n)^3)$ span.

```
(* Msort : tree -> tree
   REQ : tree
   ENS : Msort(T) returns a sorted tree containing exactly the elements of T
*)
fun Msort (Empty : tree) : tree = Empty
  | Msort (Node(l, x, r)) = Ins(x, Merge(Msort l, Merge r))
```

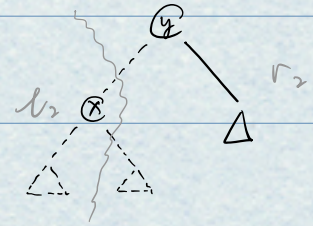


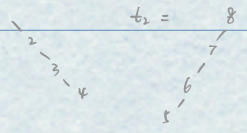
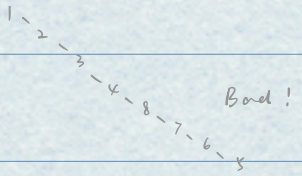

```
(* Merge : tree * tree -> tree
   REQ : t1 * t2 are sorted
   ENS : Merge(t1, t2) returns a sorted tree consisting of exactly all the elements t1 and t2 together
*)
fun Merge (Empty : tree, t2 : tree) : tree = t2
```

t_1 t_2

```
| Merge (t1, Empty) = t1
| Merge (Node(l1, x, r1), t2) =
    let
      val (l2, r2) = splitAt (x, t2)
    in
      Node (Merge (l1, l2), x, Merge (r1, r2))
```

Figure where x would belong in t_2 & split the tree t_2 there

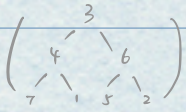


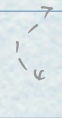
eg. t_1 =  t_2 = ⇒  Bad!

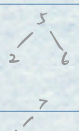
```
(* splitAt : int * tree → tree * tree
   REQ : T is sorted
   ENS : splitAt (x, T) ≙ (t1, t2) s.t. t1 & t2 together contain exactly all the elements of T and
         every element in t1 is LESS / EQUAL to x & every element in t2 is GREATER / EQUAL to x
*)
fun splitAt (x: int, Empty : tree) : tree * tree = (Empty, Empty)
  | splitAt (x, Node (l, y, r)) =
      (case Int.compare (x, y) of
         LESS ⇒ let
                  val (t1, t2) = splitAt (x, l)
                in
                  (t1, Node (t2, y, r))
                end
       | _ ⇒ let
               val (t1, t2) = splitAt (x, r)
             in
               (Node (l, y, t1), t2)
             end)
```

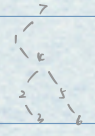
$S_{splitAt}(d) = c + S_{splitAt}(d-1)$

$S_{splitAt}(d) = O(d)$

eg. $Msort$  ⇒ Ins (3, Merge (Msort  , Msort ))

Msort  ⇒ Ins (4, Merge (Msort ②, Msort ①)) ⇒ 

Msort  ⇒ Ins (6, Merge (Msort ⑤, Msort ②)) ⇒ 

Ins (3, Merge (Msort  , Msort )) ⇒ 

Span on Merge

$S_{merge}(d_1, d_2) = c_0 + S_{splitAt}(d_2) + \max\{S_{merge}(d-1, d_2), S_{merge}(d', d_2')\}$ with $d' \le d_1 - 1$, d_2 & d_2' are the depths of l_2 & r_2 all are bounded by d_2

$= c_0 + \underbrace{S_{splitAt}(d_2)}_{\le c_2 d_2} + S_{merge}(d_1-1, d_2)$

$\le c_0 + c_2 d_2 + S_{merge}(d_1-1, d_2)$

$\le c_0 + c_2 d_2 + c_0 + c_2 d_2 + S_{merge}(d_1-2, d_2)$

$= d_1 \cdot c_0 + d_1 \cdot c_2 d_2$

$= O(d_1 d_2)$

Span in Msort

$$S_{Msort}(d) = C + \max\{S_{Msort}(d-1), S_{Msort}(d')\} + \underbrace{S_{Merge}(d_1, d_2)}_{\leq cd^2} + \underbrace{S_{INS}(d_3)}_{\leq cd}$$

$$\leq S_{Msort}(d-1) + cd^2 + cd$$

$$= S(d^3)$$

Since we are balancing the tree $O((\log n)^3)$

Ins (T) has span $S_{INS}(d)$ in $O(d)$

SplitAt (T) has span $S_{SplitAt}(d)$ in $O(d)$

Merge (T) has span $S_{Merge}(d_1, d_2)$ in $O(d_1 d_2)$

Msort (T) has span $S_{Msort}(d^3)$ in $O((\log n)^3)$

Midterm: Work in Span (every step)