作业二

1.证明：For all L:int list, msort(L) = a <-sorted permutation of L

```
fun msort [ ] = [ ]
  | msort [x] = [x]
  | msort L =  let
                 val (A, B) = split L
               in
                 merge (msort A, msort B)
               end
```

证明：

1. 当|L|= 0 或 1 时，显然成立；
2. 假设对|L|=k 和 k - 1 时成立，即当|L| = k 或 k - 1时，可以通过msort得到L的有序变形，则：
3. 当|L| = 2k时，根据val (A, B) = split L可知，|A|=|B| = k，根据假设可知，msort A和msort B均成立，又根据merge(msort A, msort B)可知，merge后得到的L是一个有序的原list的变形。
4. 当|L| = 2k - 1时，根据val (A, B) = split L可知，|A|=k，|B| = k - 1，根据假设可知，msort A和msort B均成立，又根据merge(msort A, msort B)可知，merge后得到的L是一个有序的原list的变形。
5. 综上所述，msort函数能够得到输入串L的有序串。

2.设P(t)表示：对所有整数y, SplitAt(y, t) = 二元组(t1, t2)，满足t1中的每一项 ≤ y 且 t2中的每一项≥ y，且 t1, t2由t中元素组成，证明：对所有有序树t，P(t)成立

```
fun SplitAt(y, Empty) = (Empty, Empty)
  | SplitAt(y, Node(t1, x, t2)) =
      case compare(x, y) of
        GREATER => let val (l1, r1) = SplitAt(y, t1)
                   in  (l1, Node(r1, x, t2))
                   end
      |     _   => let val (l2, r2) = SplitAt(y, t2)
                   in  (Node(t1, x, l2), r2)
                   end
```

定理：对所有树t和整数y, SplitAt(y, t) = 二元组(t1, t2), 满足depth(t1) ≤ depth t 且 depth(t2) ≤ depth t

证明：

1. 当t为Empty时，显然可知成立。
2. 当t的树高为1时，树可表示为Node(Empty, x, Empty)，SplitAt函数比较x和y的大小，若$x \leq y$，则得到Node(Empty, x, Empty)和Empty，否则得到Empty和Node(Empty, x, Empty)。可知P(t)成立。
3. 假设当t的树高为k时成立。则当t的树高为k+1时：令树根节点表示为Node(L, x, R)，根据SplitAt函数可知，
    1. 当x > y 时，执行SplitAt(y, L)，由于左子树L树高为k，根据假设可知，函数能将L分割成l1和r1，其中l1中的元素均小于y，r1中的元素均大于y。又根据函数可知，将r1并入右子树，得到新的右子树R1，其中的元素全部大于y。由于$\forall x \in t, x \in l1 \ or \in R1, l1 \cap R1 = \emptyset$，故可知成立。

2. 当x $\leq$ y时，执行SplitAt(y,R)，由于左子树R树高为k，根据假设可知，函数能将R分割成l1和r1，其中l1中的元素均小于y，r1中的元素均大于y。又根据函数可知，将l1并入左子树，得到新的左子树L1，其中的元素全部小于y。由于$\forall x \in t, x \in r1 \ or \in L1, r1 \cap L1 = \emptyset$，故可知成立。

4. 综上可得，对有序树t，P(t)成立。

3. 分析以下函数或表达式的类型

解：

```
fun all (your, base) =
    case your of 0 => base
               | _ => "are belong to us" :: all(your - 1, base)
```

根据匹配中的“0”可知，your的类型为int。根据"are belong to us"可知，返回类型为string list，由于list中的属性相同，因此base的类型也是string。故函数类型为：int * string -> string list

```
fun funny (f, []) = 0
  | funny (f, x::xs) = f(x, funny(f, xs))
(fn x => (fn y => x)) "Hello, World!"
```

由于x类型可以是多种，故认为是'a。根据第一种匹配的结果，可知函数结果类型为int，因此函数f的类型为：'a * int -> int，funny的类型为('a * int -> int) * 'a list -> int 。又根据最后一行f的定义，可知f的类型实际为：'a * int ->('a ->'b -> 'a) -> sting -> int。故最终funny的类型为：('a * int ->('a ->'b -> 'a) -> sting -> int) * 'a list -> int

4.给定一个数组A[1..n]，前缀和数组PrefixSum[1..n]定义为： PrefixSum[i] = A[0]+A[1]+...+A[i-1];

> 试编写：
>
> (1)函数PrefixSum: int list -> int list，
>
> 要求：*W**PrefixSum*(*n*) = O(n^2)。(n为输入int list的长度)
>
> (2) 函数fastPrefixSum: int list -> int list，
>
> 要求： *W**fastPrefixSum*(*n*) =*O(n)*.
>
> （提示：可借助帮助函数PrefixSumHelp）

解：

函数实现如下图所示：

```
fun PrefixSum [] = []
  | PrefixSum L:int list =
    let
      fun helpfun([], sum):int list = []
        | helpfun(x::L, sum) = (x + sum)::helpfun(L, x + sum)
    in
      helpfun(L, 0)
    end

fun fastPrefixSum [] = []
  | fastPrefixSum L =
    let
      fun helpfun([], sum, newlist) = newlist
        | helpfun(x::L, sum, newlist) = helpfun(L, x+ sum, newlist @ [x + sum])
    in
      helpfun(L, 0, [])
    end
```

5.编写函数treecompare, SwapDown 和heapify

> 一棵minheap树定义为：
>
> 1. t is Empty;
> 2. t is a Node(L, x, R), where R, L are minheaps and values(L), value(R) >= x (value(T)函数用于获取树T的根节点的值)
>
> 编写函数treecompare, SwapDown 和heapify： treecompare: tree * tree -> order (* when given two trees, returns a value of type order, based on which tree has a larger value at the root node *)
>
> SwapDown: tree -> tree (* REQUIRES the subtrees of t are both minheaps * ENSURES swapDown(t) = if t is Empty or all of t's immediate children are empty then * just return t, otherwise returns a minheap which contains exactly the elements in t. *)
>
> heapify : tree -> tree (* given an arbitrary tree *t*, evaluates to a minheap with exactly the elements of *t*. *)
>
> 分析SwapDown 和heapify两个函数的work和span。

解：

代码实现如下：

```
19
20  (* treecompare: tree * tree -> order *)
21  fun treecompare (Empty, Node(_, x, _)) = LESS
22    | treecompare (Node(_, x, _), Empty) = GREATER
23    | treecompare (Empty, Empty) = EQUAL
24    | treecompare (Node(_, x, _), Node(_, y, _)) =
25      case Int.compare(x,y)
26          of LESS => LESS
27          | EQUAL => EQUAL
28          | GREATER => GREATER
29
30  fun changeorder (l,m,r) =
31      let
32          fun cmp (x,y):int =
33              case Int.compare(x,y)
34              of GREATER => 1
35              | _ => 0
36      in
37          case (cmp(m,l), cmp(m,r), cmp(l,r))
38          of (0, 0, _) => (l,m,r)
39          | (0, 1, _) => (l,r,m)
40          | (1, 0, _) => (m,l,r)
41          | (1, 1, 0) => (m,l,r)
42          | (1, 1, 1) => (l,r,m)
43      end
44
45  (* SwapDown: tree -> tree *)
46  fun SwapDown Empty = Empty
47    | SwapDown (Node(Empty, x, Empty)) = Node(Empty, x, Empty)
48    | SwapDown (Node(Node(ll, L, lr), x, Empty)) =
49      (case Int.compare(L,x)
50          of LESS => Node(Node(ll,x,lr),L,Empty)
51          | _ => Node(Node(ll, L, lr), x, Empty))
52    | SwapDown (Node(Empty, x, Node(rl,R,rr))) =
53      (case Int.compare(R,x)
54          of LESS => Node(Empty, R, Node(rl,x,rr))
55          | _ => Node(Empty, x, Node(rl,R,rr)))
56    | SwapDown (Node(Node(ll, L, lr), x, Node( rl, R, rr ))) =
57      let
58          val (l,m,r) = changeorder(L,x,R)
59      in
60          Node((Node(ll, x, lr), m, Node( rl, r, rr )))
61      end
62
63  (* heapify : tree -> tree  *)
64  fun heapify Empty = Empty
65    | heapify (Node(Empty, x, Empty)) = Node(Empty, x, Empty)
66    | heapify (Node(Node(ll, L, lr), x, Empty)) =
67      (case Int.compare(L,x)
68          of LESS => Node(heapify(Node(ll, x, lr)), L, Empty)
69          | _ => Node(heapify(Node(ll, L, lr)), x, Empty))
70    | heapify ( Node( Empty, x, Node( rl, R, rr ) ) ) =
71      (case Int.compare(R,x)
72          of LESS => Node(Empty, R, heapify(Node(rl, x, rr)))
73          | _ => Node(Empty, x, heapify(Node(rl, R, rr))))
74    | heapify (Node(Node(ll, L, lr), x, Node( rl, R, rr ))) =
75      let
76          val (l,x,r) = changeorder(L,x,R)
77      in
78          Node(heapify(Node(ll,l,lr)), x, heapify(Node(rl, r, rr)))
79      end
80
```

对SwapDown:$\text{work} = \text{span} = O(1)$

对heapify：$\text{work} = O(2^n), \text{span} = O(n)$，其中n为树高。