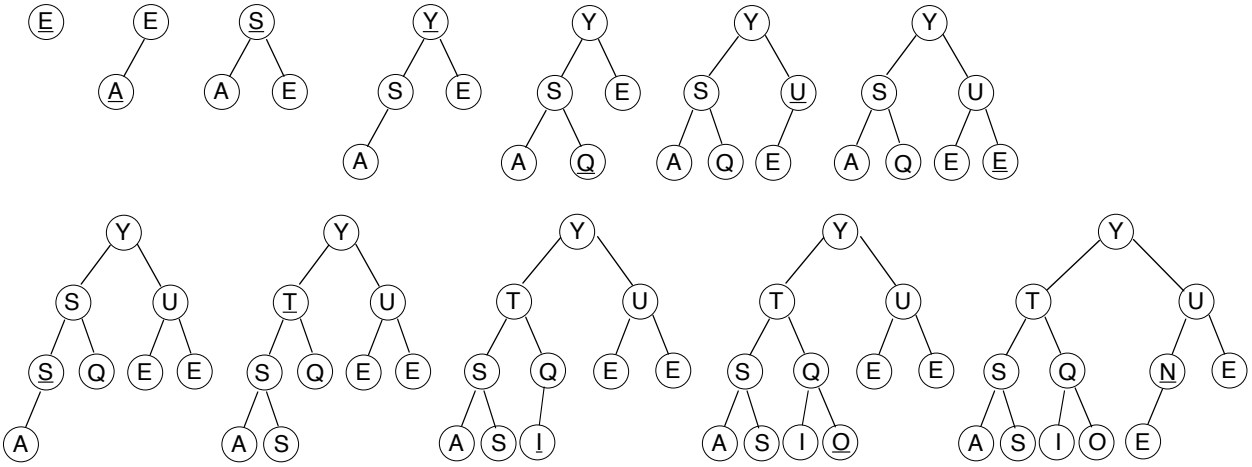
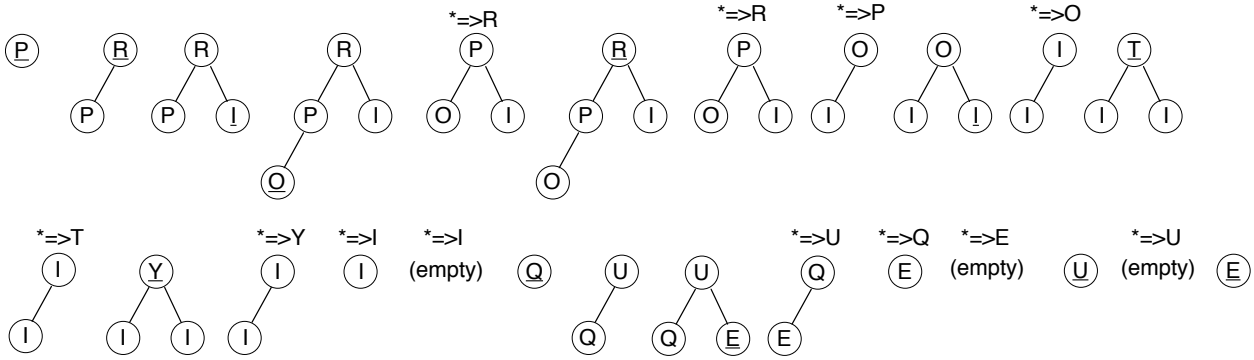
Homework for Notes 5 – 7

1. 9.21



2. 9.22



3. 9.28

Heapsort is not stable and cannot be made stable by an easy tie-breaking rule (as opposed to insertion-sort, for example.). As an example, suppose that the array elements are

5 3 3 2

Which already form a heap.

First 5 is put at the end of the array, and then one of the two children of 5 has to become the root element, to be put into the second-to-last position. This must be the right child (i.e the maximum child in case of a tie is to be the right child), in

order to obtain a stable sorting method (otherwise, the two 3's would exchange their original order).

But now consider the original sequence

6 5 3 3 2

Which also forms a heap. Here, the two children of 6 are 5 and 3, and the two children of 5 are the second 3 and 2. Then after extraction of 6, we get 5 at the root of the heap, and 5 is succeeding by the originally second 3. This produces the identical heap to the previous case, but the two 3's are in reverse position, so the rule that determines the maximum must violate stability one way or the other.

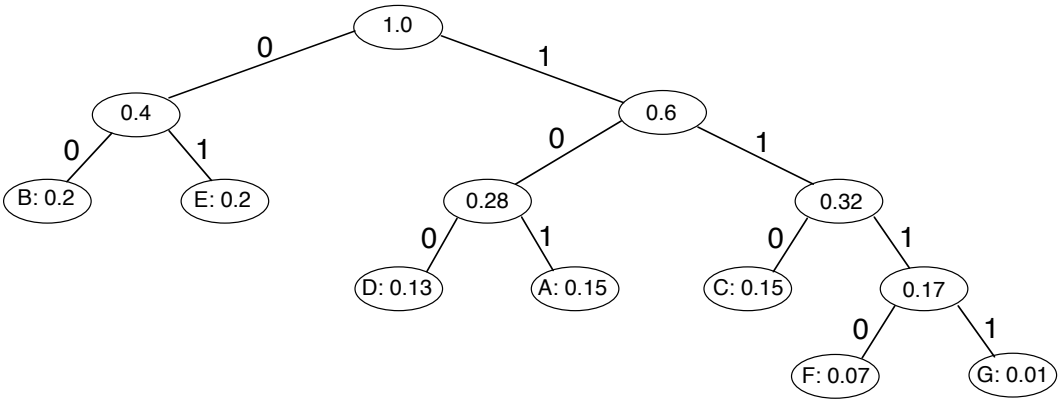
---

4. 9.30

```
Class Sort
{
    static void sort ( ITEM [] a,  int l, int r)
    {       HeapSort(a,l,r);     }
    static void HeapSort ( ITEM [] a,  int l, int r)
      {
    int k;
    PQ     pq = new PQ( r – l +1);
    For ( k = r/2; k >= 1; k --)
     pq.sink (k,r);
    while (r>1)
          {pq.exch(1,r) ;  pq.sink(1 , --r) ; }
      }
```

```
}
```

5.



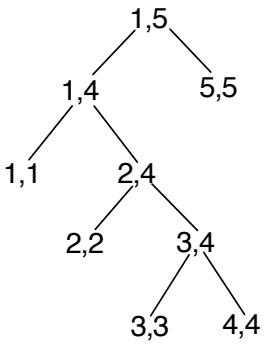
Expected number of bits = 2.77

6. The activities are first sorted by increasing finish times:

Activity	Start	Finish	
A	1	4	Include
B	3	8	Omit, overlaps A
F	6	9	Include
E	5	10	Omit, overlaps F
C	7	12	Omit, overlaps F
H	13	14	Include
G	2	15	Omit, overlaps A
D	11	16	Omit, overlaps C

7. Use program optMM.c

```
Compute c[1][5]
  k=1 gives cost 234=c[1][1]+c[2][5]+p[0]*p[1]*p[5]
  k=2 gives cost 288=c[1][2]+c[3][5]+p[0]*p[2]*p[5]
  k=3 gives cost 396=c[1][3]+c[4][5]+p[0]*p[3]*p[5]
  k=4 gives cost 216=c[1][4]+c[5][5]+p[0]*p[4]*p[5]
  c[1][5]==216,trace[1][5]==4
```



8. Computed using LCS.c

```
LCS is 01210, length==5
       2  1  0  2  1  0  2  1  0
    0  0  0  0  0  0  0  0  0  0
0   0  0  0  1  1  1  1  1  1  1
1   0  0  1  1  1  2  2  2  2  2
2   0  1  1  1  2  2  2  3  3  3
0   0  1  1  2  2  2  3  3  3  4
1   0  1  2  2  2  3  3  3  4  4
2   0  1  2  2  3  3  3  4  4  4
0   0  1  2  3  3  3  4  4  4  5
1   0  1  2  3  3  4  4  4  5  5
2   0  1  2  3  4  4  4  5  5  5
```

9. Computed using LIS.java

```
  i   y   C   j
---------------
  1   1   1   0
  2   3   2   1
  3   2   2   1
  4   5   3   3
  5   4   3   3
  6   1   2   1
  7   3   3   6
  8   4   4   7
  9   5   5   8
 10   2   3   6
  i bsTabC bsTabI
-----------------
  1       1       1
  2       1       6
  3       2      10
  4       4       8
  5       5       9
Length of LIS is 5
LIS reversed
5
4
3
1
1
```

10. Computed using LSIS.java

```
  i   y   C   j
---------------
  1   1   1   0
  2   3   2   1
  3   2   2   1
  4   5   3   3
  5   4   3   3
  6   1  -1  -1
  7   3   3   3
  8   4   4   7
  9   5   5   8
 10   2  -1  -1
  i bsTabC bsTabI
-----------------
  1       1       1
  2       2       3
  3       3       7
  4       4       8
  5       5       9
Length of LSIS is 5
LSIS reversed
5
4
3
2
1
```

11. Computed using subsetSum.java

```
  i   S
-------
  0   0
  1   3
  2   4
  3   7
  4   8
  5   9
  i   C
-------
  0   0
  1  -1
  2  -1
  3   1
  4   2
  5  -1
  6  -1
  7   2
  8   4
  9   5
 10   3
 11   3
 12   4
```

```
 13   5
 14   3
 15   4
 16   5
Solution
  i   S
-------
  5   9
  2   4
  1   3
```

12. Computed using wis.bs.c

```
  i   s   f   v   p   M
  1   1   4   1   0   1
  2   2   5   5   0   5
  3   5   6   3   2   8
  4   4   7   8   1   9
  5   7   9   1   4  10
  6   8  11   2   4  11
  7  10  15   3   5  13
  8  14  17   1   6  13
  9  14  20   4   6  15
 10  19  25   1   8  15
Include interval 9
Include interval 6
Include interval 4
Include interval 1
sum is 15
```

13. Since the item with size 13 has the highest val/size ratio, the greedy solution uses three of this item to achieve a sub-optimal solution of 42 with 7 remaining units in the knapsack.

14. Computed using knapsackTypeRS.java

```
Maximum for 47 is 50
  i size val
------------
  0   10  11
  1   14  13
  2   16  17
  3   20  19
  i max size val
----------------
 11  11   10  11
 13  11   10  11
 15  13   14  13
 17  17   16  17
 19  17   16  17
 21  22   10  11
 23  22   10  11
 27  28   10  11
```

```
 31  33   10  11
 33  34   16  17
 37  39   10  11
 47  50   10  11
Solution has value 50:
  i size  val
-------------
  1   10   11
  2   10   11
  3   10   11
  4   16   17
Unused capacity 1
```