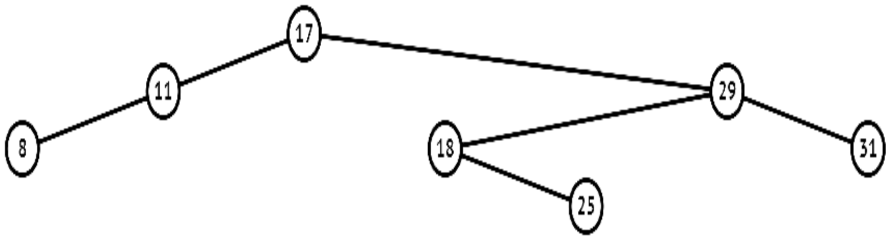
Data Structure and Algorithms 2
Homework #2 Solutions

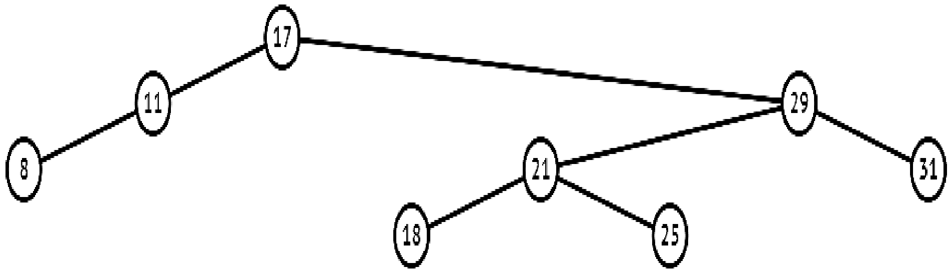
Marking Scale

Exercises	Questions	Marks	Observations
Exercise 1 (3.5 marks)	Insertion 21 Insertion 14 Insertion 20 Insertion 19	**1** **0.5** **1** **1**	
Exercise 2 **(6.5 marks)**	a) Insertion b) Deletemin c) Deletemin d) - Comparainson heap(min,max) - Generalize insertion comparaison (min ,max) d) Deletemin comparaison (min,max)	**1.5** **0.5** **0.25** **(0.25 ; 0.5)** **(0.25 ;0.5)** **(0.25 ;0.5)**	For All values -0.25 if not explained -0.25 if not explained
	e) -deleting the min - Moving - max - deletemin	 **0.5** **0.5** **0.5** **0.5**	
Exercise 3 **(4 marks)**	a) Linear probing Table1(homeslot,probing seq) Table2(hash table)	 **0.75** **0.25**	Tabla1 : 0.25 indexes without **collision** 0.5 indexes with **collision resolution**
	b) Quadratic Probing Table1(homeslot,probing seq) c) Table2(hash table)	 **1.25** **0.25**	Table1 : 0.25 indexes without **collision** **1** indexes with **collision resolution**
	d) Double hashing Table1(homeslot,probing seq) Table2(hash table)	 **1.25** **0.25**	Table1 : 0.25 indexes without **collision** 1 indexes **collision resolution**
Exercise 4 **(6 marks)**			

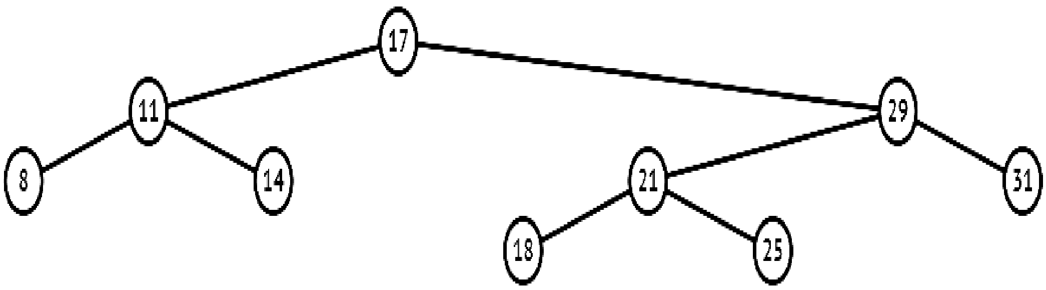
Exercise 1 (AVL tree)



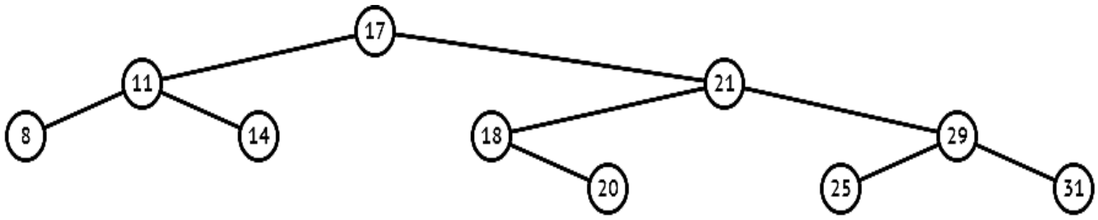
Insertion of 21



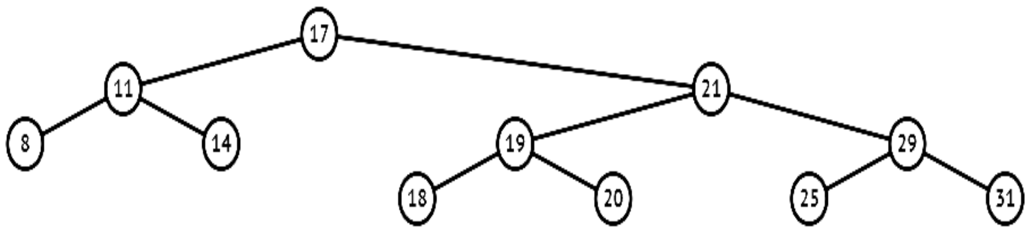
Insertion of 14



Insertion of 20

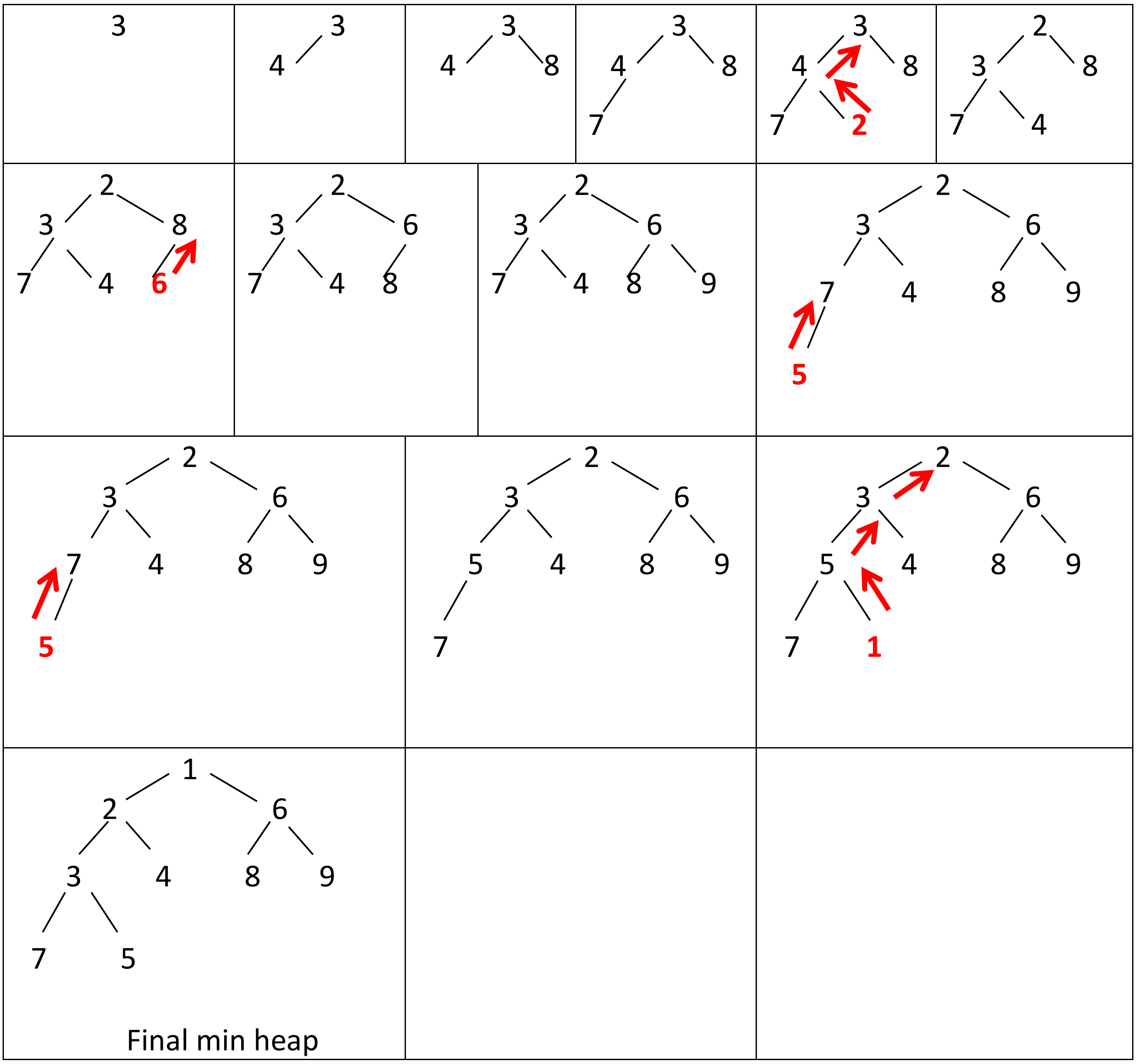


Insertion of 19

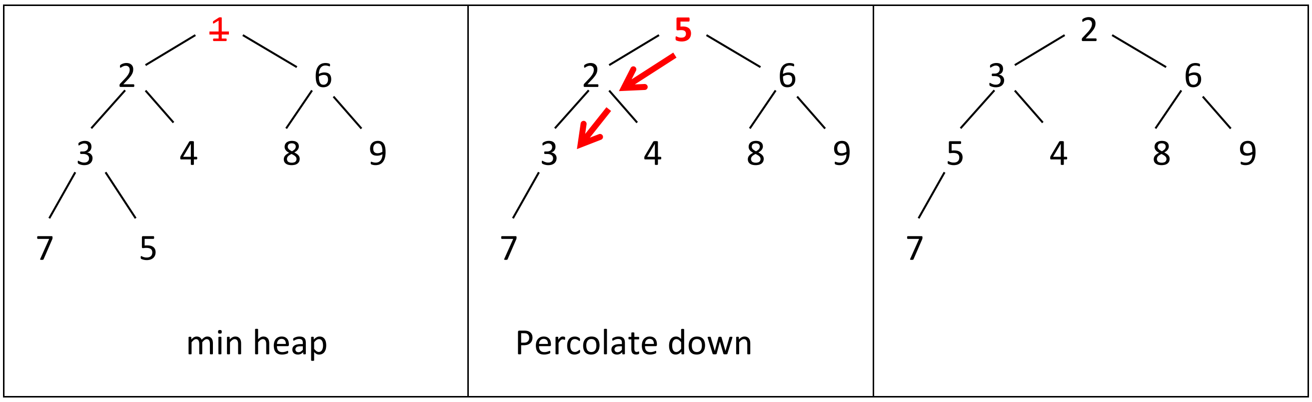


Exercise 2 (Binary heaps)

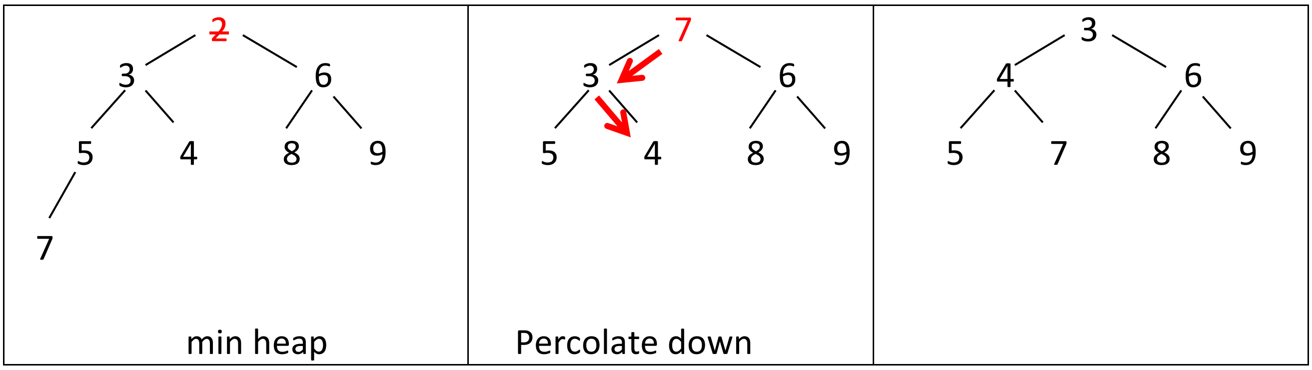
a) **Insertion :** insert 3,4,8,7,2,6,9,5,1.



b) **Deletemin :** 1 call deletemin



c) **Deletemin :** 2 call deletemin



d) Given the binary min heap obtained after answering question (c), what are the minimum and maximum numbers of comparisons one might have to do when inserting the next value?

- Minimum number of comparisons **= 1** (compare with the parent and new value > parent value : priority order guaranted)
- Maximum number of comparisons **= 3** (compare with the parent the n its parents in high levels till the root : new value < parent value and less then all values till the root)

Generalize your answer for a min heap of height h.

Given a min heap of height h The minimum and maximum number of comparisons to do when inserting the next value are :

Minimum = 1

Maximum **: h (or h+1)**

Case1 : Min Heap is complete of height h : insertion of a new value needs h+1 maximum comparaisons .
Case2 : Min Heap of height h and it is not complete : insertion of a new value needs h maximum comparaisons

- Given a min heap of height h The minimum and maximum number of comparaisons to do when doing a deleteMin are :
  - Minimum = **2** (one comparaison of the last value with the left child and the comparaison between left and right child to get the min).
- Maximum =**2 h** (each level we have 2 comparaisons)

g)

- **Deleting the minimum value in a binary min heap of size N** this is just a deletemin operation : finding the minimum is O(1) because it is always at the root (position 1) . put the right most value in the last level at the root and percolate down wich costs at the worst case the height of the AVl tree since the height of an complete binary tree is O(Log(N)), the total runtime **is O(logN)** .
- **Moving the values from min heap into an initially empty array of the same size. The final contents of the array should be sorted from low to high** call of deletemin N times : each call for deletemin gives the next value from (low to high) which is log(N), Write it to next empty location is constant time . For N operation the total cost is **Nlog(N)**.
- **Finding the maximum value in a binary min heap of size N** The maximum will be in the last (N/2) values. you can just search this but it still **O(N)**.
- **Deletemin from a priority Queue implemented with a binary min heap** Deletemin : remove root and remplate it with lastnalue then percolate down till the order ptoperty is guaranted it may take the height of the heap thus **log(N)**

Exercise 3 (Hashing)

hash table consisting of **M = 11** slots, hash function h1() :

int h1 (int key)

```
{ int x = (key + 7) * (key + 7);
x = x / 16;
x = x + key;
x = x % 11;
return x; }
```

1) **Linear probing**

   Index(key) = (h1(key) +f(i)) mod 11 with the probe function : f(i) = i

Key= 43 : Index =h1(43) = 1 Key= 23 : Index =h1(23) = 2 Key= 1 : Index =h1(1) = 5 Key= 0 : Index =h1(0) = 3 Key= 15 : Index =h1(15) = 1 collision 2 ,3 collision index = 4	Key= 31 : Index =h1(31) = 0 Key= 4 : Index= h1(4) = 0 1,2,3,4,5, : collision index =6 Key= 7 : Index =h1(7) = 8 Key= 11 : Index =h1(11) = 9 Key= 3 : Index =h1(3) = 9 collision (i=1) : index = 9+((1^2+1)/2 =0+1=10 Index = 10

Table 1 (Linear Probing)

Key Value	Home Slot	Probe Sequence
43	1	
23	2	
1	5	
0	3	
15	1	2, 3, 4
31	0	
4	0	1, 2, 3, 4, 5, 6
7	8	
11	9	
3	9	10

Table 2 hash table (Linear Probing)

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	31	43	23	0	15	1	4		7	11	3

2) Quadratic probing

Index(key) = (h1(key) +f(i)) mod 11 with the probr function : $f(i) = (i^2+i) /2$

Key= 43 : Index =h1(43) = 1 Key= 23 : Index =h1(23) = 2 Key= 1 : Index =h1(1) = 5 Key= 0 : Index =h1(0) = 3 Key= 15 : Index =h1(15) = 1 collision (i=1) : index = $1+((1^2+1)/2$ =2 collision (i=2) : index = $1+((2^2+2)/2$ =1+3=4 index = 4	Key= 31 : Index =h1(31) = 0 Key= 4 : Index= h1(4) = 0 (i=1) : index = $0+((1^2+1)/2$ =1 collision (i=2) : index = $0+((2^2+2)/2$ =0+3=3 collision (i=3) : index = $0+((3^2+3)/2$ =0+6=6 index =6 Key= 7 : Index =h1(7) = 8 Key= 11 : Index =h1(11) = 9 Key= 3 : Index =h1(3) = 9 collision Index = 10

Table 1 (Quadratic probing)

Key Value	Home Slot	Probe Sequence
43	1	
23	2	
1	5	
0	3	
15	1	2, 4
31	0	
4	0	1, 3, 6
7	8	
11	9	
3	9	10

Table 2 Hash table (Quadratic probing)

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	31	43	23	0	15	1	4		7	11	3

3) Double hashing

the secondary hash function Reverse(key), which reverses the digits of the key and returns that value; for example, Reverse(7823) = 3287.

Index(key) = (h1(key)+ i *h2(key))mod 11

Key= 43 : Index =h1(43) = 1 Key= 23 : Index =h1(23) = 2 Key= 1 : Index =h1(1) = 5 Key= 0 : Index =h1(0) = 3 Key= 15 : Index =h1(15) = 1 collision h2(15)= reverse (15) = 51 (i=1) : Index = (1+ 1*52)%11 = 8 Key= 31 : Index =h1(31) = 0 Key= 4 : Index= h1(4) = 0 collision h2(4)=reverse (4) = 4 (i=1) : index = (0+(1*4)) %11=4 index =4	Key= 7 : Index =h1(7) = 8 collision h2(7)=reverse (7) = 7 (i=1) : index =(8+(1*7)) %11 =4 collision (i=2) : index =(8+(2*7)) %11 =0 collision (i=3) : index =(8+(3*7)) %11 =7 index =7 Key= 11 : Index =h1(11) = 9 Key= 3 : Index =h1(3) = 9 collision h2(3)=reverse (3) = 3 (i=1) : index =(9+(1*3)) %11 =1 collision (i=2) : index =(9+(2*3)) %11 =4 collision (i=3) : index =(9+(3*3)) %11 =7 collision (i=3) : index =(9+(4*3)) %11 =10 index =10

Table 1 (Double Hashing)

Key Value	Home Slot	Probe Sequence
43	1	
23	2	
1	5	
0	3	
15	1	8
31	0	
4	0	4
7	8	4, 0, 7
11	9	
3	9	1, 4, 7, 10

Table 2 hash table (Double Hashing)

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	31	43	23	0	4	1		7	15	11	3

Exercise 4

1.

```
Type TBook = Structure
   Title, Author : string ;
   Next : Pointer ( TBook )
End ;

Type TCategory = Structure
   CategoryName : string ;
   BookCount : integer ;
   HeadBookList : Pointer ( TBook )
   Next : Pointer ( TCategory ) ;
End ;

Var
   Library : Pointer ( TCategory ) ;

Procedure Initialization ;
Begin
   Library ← Nil ;
End ;
```

—-----------------------

2.

```
Procedure AddCategory ( CatName : string ) ;
Var    P, PP, Q : Pointer ( TCategory ) ;
Begin
   P ← Library ;
   PP ← Nil ;
   While P ≠ Nil and CategoryName(P) ≠ CatName Do
      PP ← P ;
      P ← Next ( P ) ;
   End While;

   If P ≠ Nil then Write ( 'This category already exists' ) ;
   Else
      Allocate( Q ) ;
      Assign_Value ( Q, CatName, 0, Nil ) ;
      Assign_Address ( Q, Nil ) ;
      If PP = Nil then Library ← P
      Else
         Assign_Address ( PP, Q ) ;
      End If;
   End ;
End ;
```

3.

```
Procedure AddBook(CatName, Title, Author: string);
Var
   P : Pointer (TCategory);
   Q : Pointer (TBook);
Begin
   P ← Library;
   While P ≠ Nil and CategoryName(P) ≠ CatName Do
      P ← Next (P);
   End While;

   If P = Nil then Write ('This category does not exist');
   Else
      Allocate (Q);
      Assign_Address(Q, HeadBookList(P));
      Assign_Value(Q, Title, Author);
      Assign_Value(P, CategoryName(P), BookCount(P) + 1, Q);
   End If;
End;
```

4.

```
Procedure Display ( CatName : string );
Var
   P : Pointer (TCategory);
   Q : Pointer (TBook);
Begin
   P ← Library;
   While P ≠ Nil and CategoryName(P) ≠ CatName Do
```

```
      P ← Next ( P );
   End While;

   If P = Nil then Write ('This category does not exist')
   Else
      Q ← HeadBookList(P);
      While Q ≠ Nil Do
         Write (Title(Q), Author(Q));
         Q ← Next (Q);
      End While;
   End If;
End;
```

5.

```
Function TotalBooks : Integer;
Var
   P : Pointer (TCategory);
   Nb : integer;
Begin
   Nb ← 0;
   P ← Library;
   While P ≠ Nil Do
      Nb ← Nb + BookCount(P);
      P ← Next (P);
   End;
   TotalBooks ← Nb;
End;
```

6.

```
Procedure DeleteCategory ( CatName : string );
Var
   P, PP : Pointer (TCategory);
   Q, Q1 : Pointer (TBook);
Begin
   P ← Library;
   PP ← Nil;
   While P ≠ Nil and CategoryName(P) ≠ CatName Do
      PP ← P;
      P ← Next ( P );
   End While;

   If P = Nil then Write ('This category does not exist')
   Else
      Q ← HeadBookList(P);
      While Q ≠ Nil Do
         Q1 ← Q;
         Q ← Next(Q);
         Free ( Q1 );
      End While;
```

```
        Assign_Address(PP, Next(P));
        Free ( P );
    End If;
End;
```