

COMP5112 Assignment 2

Shuyu Zhao (shuyu.zhao@connect.polyu.hk)

October 3, 2020

Problem 1.

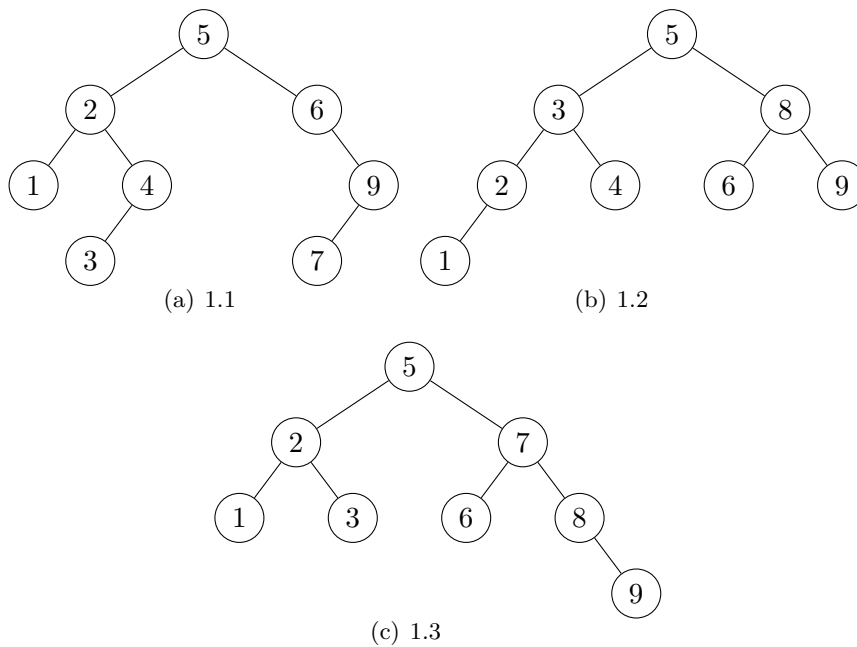


Figure 1: Problem 1

Problem 2.

- (a) A isn't a valid BST. (6 is small than 7 while 6 is on 7's right subtree)
- (b) B is a valid BST. Resultant Tree see Fig 2(a).
- (c) C is a valid BST. Resultant Tree see Fig 2(b).

(d) D is a valid BST. Resultant Tree see Fig 2(c).

(e) E is a valid BST. Resultant Tree see Fig 2(d).

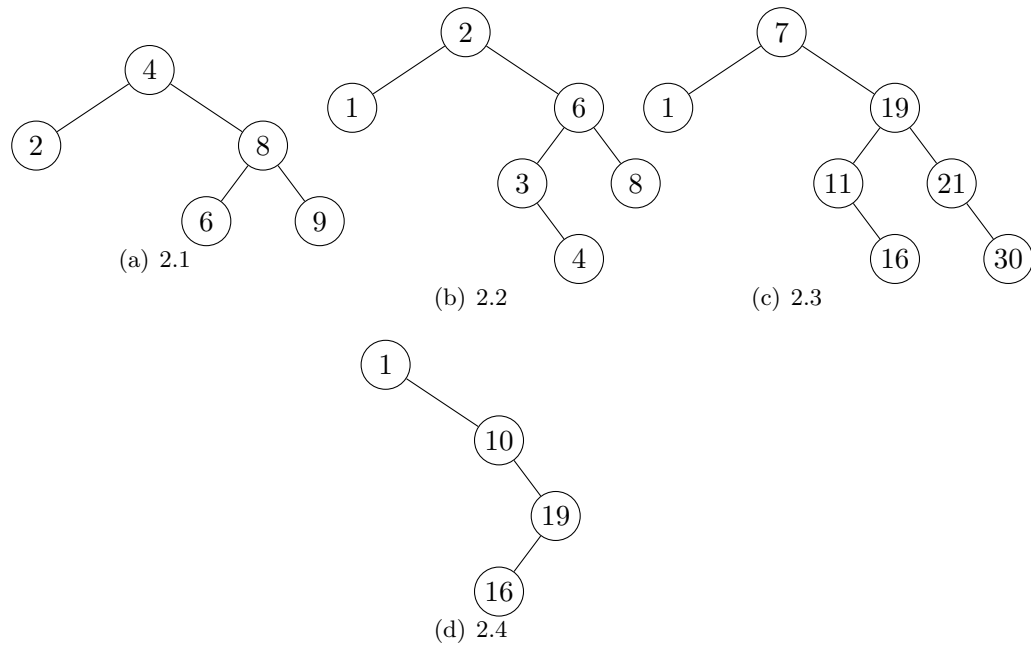


Figure 2: Problem 2

Problem 3.

Answer is **2**.

Problem 4.

My approach: put values in the binary tree to a list, then sort the list, do inorder traversal to the binary tree to put value into it finally. See pseudocode in Algorithm 1.

The time complexity of this approach is decided by sorting algorithm. The minimum time complexity of this approach is $\mathcal{O}(n \log n)$ if we use quick sort. In general cases, the time complexity will be $\mathcal{O}(n \log n)$, however, the time complexity will be $\mathcal{O}(n^2)$ in the worst case (When the call tree of quick sort is the most unbalanced).

```

Input:  $T$ , A normal binary Tree
Output:  $T$ (in place editing)
1 Create an empty list  $L$ ;
2 Function Store_to_list( $T, L$ ):
3   if  $T = NULL$  then
4     return;
5   end
6   Store_to_list( $T.left, L$ );
7   append  $T.data$  to  $L$ ;
8   Store_to_list( $T.right, L$ );
9 Function Partition( $L, p, r$ ):
10   $x = L[r]$ ;
11   $i = p - 1$ ;
12  for  $j = p$  to  $j = r - 1$  do
13    if  $L[j] \leq x$  then
14       $i = i + 1$ ;
15      exchange  $L[i]$  and  $L[j]$ ;
16    end
17  end
18  exchange  $L[i + 1]$  and  $L[r]$ ;
19  return  $i + 1$ ;
20 Function Quick_Sort( $L, p, r$ ):
21  if  $p < r$  then
22     $q = \text{Partition}(L, p, r)$ ;
23    Quick_Sort( $L, p, q - 1$ );
24    Quick_Sort( $L, q + 1, r$ );
25  end
26 Create a temporary variable  $flag = 0$ ;
27 Function Convert_to_BST( $T, L$ ):
28  if  $T = NULL$  then
29    return;
30  end
31  Convert_to_BST( $T.left, L$ );
32   $T.data = L[flag]$ ;
33   $flag = flag + 1$ ;
34  Convert_to_BST( $T.right, L$ );

```

Algorithm 1: Pseudocode for converting a binary tree to a BST

Problem 5.

If we choose one key as root, so it will have four situations:

1. 0 key on left and 3 keys on right
2. 3 keys on left and 0 key on left
3. 1 key on left and 2 keys on right
4. 2 keys on left and 1 key on right

Hence, problem converts to how many different BST could created by 0/1/2/3 keys.

1. With 0 key and 1 key, we can only create 1 type of BST.
2. With 2 keys, we choose one key as root, so it will have 2 type of BST.
3. With 3 keys, if we choose one key as root, so it will have three situations:
 - (a) 0 key on left and 2 keys on right, result to 2 types
 - (b) 2 keys on left and 0 keys on right, result to 2 types
 - (c) 1 key on left and 1 key on right, result to 1 type

So, we will have $2(2 + 2 + 1 + 2) = 14$ types of BST with 4 keys in total. In general, we will have $C_n = \frac{(2n)!}{(n+1)!n!}$ types of BST with n distinct keys.

Problem 6.

1. **True**, in-order traversal always start from the left-most child(the smallest one) and end on the right-most child(the biggest one), which is an increasing order.
2. **False**, we cannot reconstruct the BST based on one single traversal. Combinations of traversal(pre-order + in-order or post-order + in-order) can help us to construct the original BST. For instance, Using in-order traversal(1,2,3) and post-order(2,1,3) can construct one specific BST(1-2[root]-3), while we can construct three BSTs(1[root]-2-3, 1-2[root]-3, 1-2-3[root]) with only in-order traversal.

Problem 7.

My approach: Do bottom-up comparison or upheap. See pseudocode in Algorithm 2. The time complexity of this approach is $\mathcal{O}(n)$. Each time we call **Heapify**, it takes $\mathcal{O}(k)$, k stands for the heap's height. Different nodes has different k and we have $\frac{n}{2^{(k+1)}}$ nodes on height k . Hence, $T(n) = \left(\sum_{k=1}^{\log(n)} \frac{1}{2^k} \times k\right) * n$. Assume $S = \sum_{k=1}^{\log(n)} \frac{1}{2^k} \times k$, $\frac{S}{2} = \sum_{k=1}^{\log(n)-1} \frac{1}{2^k} \times k$, $S - \frac{S}{2} = \frac{S}{2} = \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^k} - \frac{1}{2^{k+1}} \times k \leq 2$. So $T(n) \leq 2n = \mathcal{O}(n)$. Demo see Fig 3.

```

Input:  $V$ , An unsorted array,  $i$ , index of the array
Output:  $V$ , Min heap(in place editing)
1  $l = \text{Left}[i];$ 
2  $r = \text{right}[i];$ 
3 Function  $\text{Heapify}(V, i):$ 
4   if  $l \leq V.\text{heap-size}$  and  $V[l] < V[i]$  then
5      $\text{min} = l;$ 
6   else
7      $\text{min} = i;$ 
8   end
9   if  $r \leq V.\text{heap-size}$  and  $V[r] < V[\text{min}]$  then
10     $\text{min} = r;$ 
11  end
12  if  $\text{min} \neq i$  then
13    exchange  $V[i]$  and  $V[\text{min}];$ 
14     $\text{Heapify}(V, \text{min})$ 
15  end
16  $V.\text{heap-size} = V.\text{length};$ 
17 for  $i = V.\text{length}\%2$  to 1 do
18    $\text{Heapify}(V, i)$ 
19 end

```

Algorithm 2: Pseudocode for converting an unsorted array to a Min heap

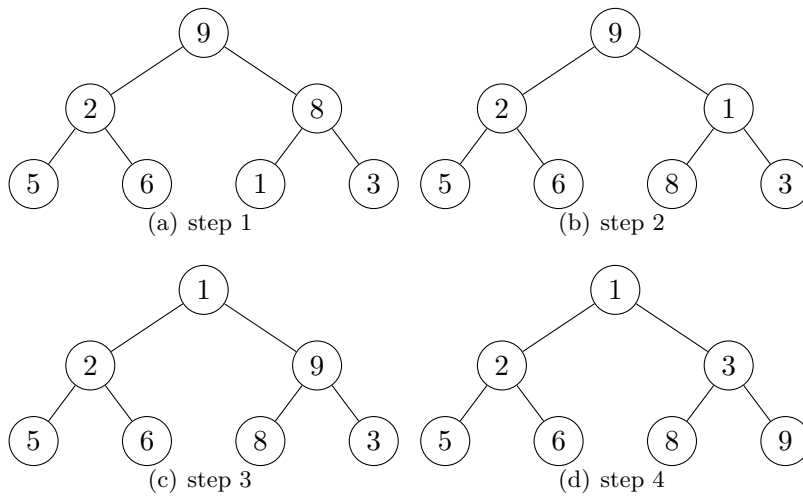


Figure 3: Problem 7

Problem 8.

The answer is {null, 2, 4, 3, 9, 7, 6}.

Problem 9.

The answer are 1,3.

Problem 10.

Answer see Fig 4, log see below.

11	39	20		16	44	88	12	23	13	94
----	----	----	--	----	----	----	----	----	----	----

Figure 4: Problem 10

```

12 Succ with hv = 7-[0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 0]
44 Succ with hv = 5-[0, 0, 0, 0, 0, 44, 0, 12, 0, 0, 0]
13 Succ with hv = 9-[0, 0, 0, 0, 0, 44, 0, 12, 0, 13, 0]
88 Fail with hv = 5-[0, 0, 0, 0, 0, 44, 0, 12, 0, 13, 0]
88 Succ with hv = 6-[0, 0, 0, 0, 0, 44, 88, 12, 0, 13, 0]
23 Fail with hv = 7-[0, 0, 0, 0, 0, 44, 88, 12, 0, 13, 0]
23 Succ with hv = 8-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 0]
94 Fail with hv = 6-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 0]

```

```

94 Fail with hv = 7-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 0]
94 Fail with hv = 8-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 0]
94 Fail with hv = 9-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 0]
94 Succ with hv = 10-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Fail with hv = 5-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Fail with hv = 6-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Fail with hv = 7-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Fail with hv = 8-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Fail with hv = 9-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Fail with hv = 10-[0, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
11 Succ with hv = 0-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Fail with hv = 6-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Fail with hv = 7-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Fail with hv = 8-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Fail with hv = 9-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Fail with hv = 10-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Fail with hv = 0-[11, 0, 0, 0, 0, 44, 88, 12, 23, 13, 94]
39 Succ with hv = 1-[11, 39, 0, 0, 0, 44, 88, 12, 23, 13, 94]
20 Fail with hv = 1-[11, 39, 0, 0, 0, 44, 88, 12, 23, 13, 94]
20 Succ with hv = 2-[11, 39, 20, 0, 0, 44, 88, 12, 23, 13, 94]
16 Succ with hv = 4-[11, 39, 20, 0, 16, 44, 88, 12, 23, 13, 94]

```

Problem 11.

Answer see Fig 5, log see below.

44	12	13	23	20	16	88	39	94	11	
----	----	----	----	----	----	----	----	----	----	--

Figure 5: Problem 11

```

12 Succ with i = 0, hv = 1-[0, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0]
44 Succ with i = 0, hv = 0-[44, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0]
13 Succ with i = 0, hv = 2-[44, 12, 13, 0, 0, 0, 0, 0, 0, 0, 0]
88 Fail with i = 0, hv = 0-[44, 12, 13, 0, 0, 0, 0, 0, 0, 0, 0]
88 Fail with i = 1, hv = 2-[44, 12, 13, 0, 0, 0, 0, 0, 0, 0, 0]
88 Succ with i = 0, hv = 6-[44, 12, 13, 0, 0, 0, 88, 0, 0, 0, 0]
23 Fail with i = 0, hv = 1-[44, 12, 13, 0, 0, 0, 88, 0, 0, 0, 0]
23 Succ with i = 0, hv = 3-[44, 12, 13, 23, 0, 0, 88, 0, 0, 0, 0]
94 Fail with i = 0, hv = 6-[44, 12, 13, 23, 0, 0, 88, 0, 0, 0, 0]
94 Succ with i = 0, hv = 8-[44, 12, 13, 23, 0, 0, 88, 0, 94, 0, 0]
11 Fail with i = 0, hv = 0-[44, 12, 13, 23, 0, 0, 88, 0, 94, 0, 0]
11 Fail with i = 1, hv = 2-[44, 12, 13, 23, 0, 0, 88, 0, 94, 0, 0]
11 Fail with i = 2, hv = 6-[44, 12, 13, 23, 0, 0, 88, 0, 94, 0, 0]
11 Fail with i = 3, hv = 1-[44, 12, 13, 23, 0, 0, 88, 0, 94, 0, 0]
11 Succ with i = 0, hv = 9-[44, 12, 13, 23, 0, 0, 88, 0, 94, 11, 0]

```

```
39 Fail with i = 0, hv = 6-[44, 12, 13, 23, 0, 0, 88, 0, 94, 11, 0]
39 Fail with i = 1, hv = 8-[44, 12, 13, 23, 0, 0, 88, 0, 94, 11, 0]
39 Fail with i = 2, hv = 1-[44, 12, 13, 23, 0, 0, 88, 0, 94, 11, 0]
39 Succ with i = 0, hv = 7-[44, 12, 13, 23, 0, 0, 88, 39, 94, 11, 0]
20 Fail with i = 0, hv = 9-[44, 12, 13, 23, 0, 0, 88, 39, 94, 11, 0]
20 Fail with i = 1, hv = 0-[44, 12, 13, 23, 0, 0, 88, 39, 94, 11, 0]
20 Succ with i = 0, hv = 4-[44, 12, 13, 23, 20, 0, 88, 39, 94, 11, 0]
16 Succ with i = 0, hv = 5-[44, 12, 13, 23, 20, 16, 88, 39, 94, 11, 0]
```

Problem 12.

1. chaining: **pros**: It's easy to deal with collision with just adding value on the front or tail of the linked list. **cons**: It has to maintain many linked list, which costs extra space. Also, It takes more time to find a value in extreme situation(like all value in one linked list).
2. linear probing: **pros**: It's easy to use and fast. **cons**: It's hard to delete value in hash table using linear probing. Marking the deleted one in array will waste the array space.
3. quadratic probing: **pros**: It will face less clustering problem than linear probing. **cons**: It's not guarantee to find a empty cell to store data when load factor becomes larger than 0.5.

Problem 13.

Quick Sort see below and Merge Sort see Fig 6.

Step 1: Choose a pivot 


Step 2: Lesser values go to the left, equal or greater values go to the right





Step 3: Repeat step 1 

Step 4: Same to step 2 

Step 5: Repeat step 1 

Step 6: Same to step 2 

Step 7: Repeat step 1 

Step 8: Same to step 2 

Step 9: Sorting finished 

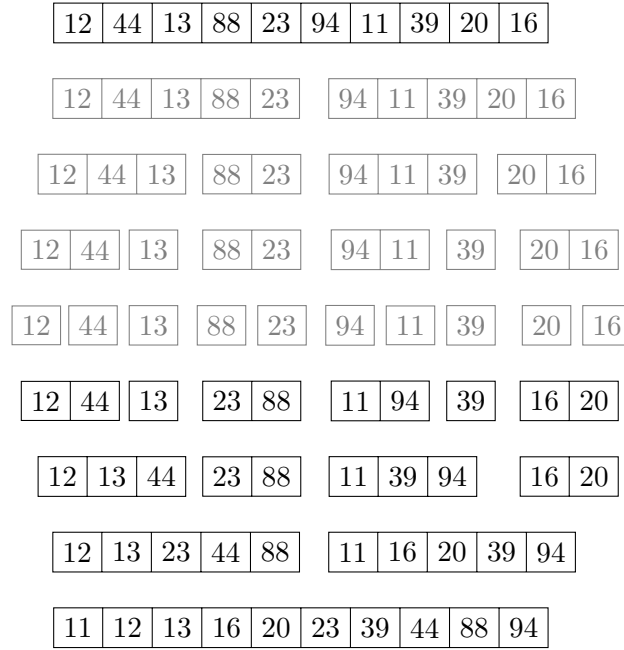


Figure 6: Problem 13: Merge Sort

Problem 14.

Insertion Sort. For insertion sort, if the array is sorted or almost sorted, the algorithm will only do comparisons from first element to the last element (n times) and a small number of swaps (constant time). Hence, the time complexity for insertion sort is $\mathcal{O}(n)$ for best case (the array is sorted or almost sorted).

Problem 15.

When the pivots are always the minimum number or the maximum number of the sequence (or sub-sequence). In this case, either left or right part will be extremely long after partition. So, the recursion tree will be totally unbalanced with height n , which result in the final time complexity is $\mathcal{O}(n^2)$.

Problem 16.

1. selection sort: **pros:** It's an in-place sorting algorithm. **cons:** In

general cases, It takes more time to sort than merge sort and quick sort.

2. insertion sort: **pros:** It's stable and it has the greatest performance when dealing with almost sorted list. **cons:** In general cases, It takes more time to sort than merge sort and quick sort.
3. merge sort: **pros:** It's a stable sorting algorithm. Also, it's fast in general cases. **cons:** It takes extra spaces to store sub-array.
4. quick sort: **pros:** It's the fastest algorithm compared with above three. **cons:** It's not stable and It will have bad performance in the worst case.

Credits

This assignment is written by L^AT_EX with template under MIT License.