

## Solutions to week5 flipped worksheet

1.
  - a. T U V
  - b. B R R B
  - c. 

0	1	2
0	0	3
0	0	3
2.
  - a. Use heapsort to sort both arrays  $a[]$  and  $b[]$ . No extra space is needed. Now compare  $a[i]$  to  $b[i]$ . If  $a[]$  and  $b[]$  are rearrangements of each other, we will have sorted  $a[]$  and  $b[]$  the same elements in all positions
  - b. Hash all  $a[]$  into a hashtable (need linear extra space). Now for each  $b[i]$  check if  $b[i]$  is in the hashtable. Since all elements are distinct, this test should tell you if elements of  $a[]$  are the same as elements of  $b[]$
3. A D X
4.
  - a. open – do not know any sub quadratic algorithm for solving 3-sum problem
  - b. Possible – We can hash  $a[]$  into a hash table. Then for each pair  $(a[i], a[j])$  look for an element  $a[k] = -(a[i]+a[j])$ . There are  $n^2$  pairs and hence we can solve 3-sum problem in quadratic time with linear extra space.
  - c. Possible. Use the amortized array implementation of a stack. We can use two such stacks, `enqueue_stack` and `dequeue_stack`. We enqueue elements (push) into `enqueue_stack` and dequeue elements of `dequeue_stack` (pop).
  - d. Possible. Since the tree is balanced and has height  $\log N$ , just go all the way to right to find the max
  - e. Impossible. If so, we can build a PQ and do `delMax` in  $2/3 N \log N$  time. This is less than the sorting lower bound.
  - f. Possible. Use Dijkstra's 3-way partitioning
  - g. Open – holy grail of sorting. Mergesort is one such thing, but need extra space
  - h. Impossible – need to traverse the tree  $k$  times to find this and  $k$  can be  $n/2$  and hence linear in worst case.
5.
  - a. Possible, Use weighted union-find
  - b. Impossible. If this is possible, we will have a sorting algorithm that satisfies  $T(N) = 2T(N/2) + \frac{1}{2} N$  and can lead to  $T(N) \sim \frac{1}{2} N \log N$  (not possible)
  - c. Impossible. If we can build a BST in  $17N$  compares, we can sort in linear time.
  - d. Possible. Use bottom-up heap construction
  - e. Possible. This is called mergesort
  - f. Possible. Do binary search to find the first instance of the key and again binary search to find the last instance of the key. Find the difference between the two indices in  $\sim 2 \log N$  time.