

Name:SOLUTIONS to SELECTED PROBLEMS.....

ID Number:

Exam Type:

...B...

Exam Number:

.....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... 7: 8: ... 9: ... Total:

Problem 1. (30 POINTS)

Problem 2. (33 POINTS)

TYPE A

(1) $T(n) = T(n/4) + n.$

Case 3 of MM, gives $T(n) = \Theta(n)$

(2) $T(n) = 9T(n/3) + n.$

Case 1 of MM, gives $T(n) = \Theta(n^2)$

(3) $T(n) = 4T(n/4) + n \lg n.$

Case 2 of MM, gives $T(n) = \Theta(n(\lg n)^2)$

TYPE B

(1) $T(n) = 16T(n/4) + n.$

Case 1 of MM, gives $T(n) = \Theta(n^2)$

(2) $T(n) = 9T(n/3) + n^3.$

Case 3 of MM, gives $T(n) = \Theta(n^3)$

(3) $T(n) = 2T(n/2) + n \lg n.$

Case 2 of MM, gives $T(n) = \Theta(n(\lg n)^2)$

Problem 3. (30 POINTS)

$$T(n) = 2T(n/2) + (n - 1), \quad T(2) = 2.$$

Use the recursion tree solution for the merge sort recurrence done in class; the only difference is that the parents of the leaves ($n/2$ of nodes) have values 2 not the $2 - 1 = 1$ in the in-class example. Thus the solution is not the $T(n) = n \lg n - (n - 1)$ derived in class but $n/2$ more i.e. $T(n) = n \lg n - n + 1 + n/2 = n \lg n - n/2 + 1$. Here is a formal proof using the iteration method.

$$\begin{aligned} T(n) &= 2T(n/2) + n - 1 = 2(2T(n/2^2) + n/2 - 1) + n - 1 \\ &= 2^2T(n/2^2) + (n - 2) + (n - 1) = 2^i T(n/2^i) + in - (2^0 + \dots + 2^{i-1}) \\ &= 2^i T(n/2^i) + in - 2^i + 1 = 2^{\lg n - 1} T(n/2^{\lg n - 1}) + (\lg n - 1) \cdot n - n/2 + 1 \\ &= (n/2)T(2) + n \lg n - n - n/2 + 1 = n \lg n - n/2 + 1 \end{aligned}$$

Problem 4. (60 POINTS)

Problem 5. (30 POINTS)

There are various ways to solve this problem... optimally. Sorting is not needed. It helps that no duplicates appear in $A[0..n-1]$. The sum of the $1..n+1$ is known (arithmetic sequence). Add all the numbers from $A[0]$ through $A[n-1]$ and subtract from that sum! It is time efficient and sort of space efficient as long sum can accommodate the sum of the first $n + 1$ natural integer numbers. You may be able to use XOR, or just negation to get something that avoids large sum values!

```

Find(A[0...n])
sum= (n+1)*(n+2)/2; nsum=0; // sum = 1 + 2 + ... + n + (n+1)
for(i=0;i<n;i++){
    nsum= nsum+A[i];
}
return(sum-nsum);

```

Problem 6. (30 POINTS)

TYPE A. The conjunction can contain no more documents than the shorter of the two lists i.e. T of $\sqrt{n}$ length. Thus do a binary search of each docID of T into J that takes $O(\lg n)$ per binary search and overall $O(\sqrt{n} \lg n)$, and determine which docIDs of T appear in J as well, and thus should be included in C . Time is $O(\sqrt{n} \lg n)$ for lines 1-5 and also 7-11. The length of C is as small as the output. The first binary search of lines 1-5 determines the size of C , which is **number** and then allocates space for C . The subsequent binary search is the one that fills C with elements that appear both in T and J . Total time is $O(\sqrt{n} \lg n)$ and space as little as needed for C !

```

AND(T,J,sqrt(n),n)
0. number = 0;
1. for(i=0;i<|T|;i++){
2.   index=BinSearch(T[i],J,n);
3.   if index != -1
4.     number++; // Count number of keys
5. }
6. C=new[number]; number = 0;
7. for(i=0;i<|T|;i++){
8.   index=BinSearch(T[i],J,n);
9.   if index != -1
10.    C[number]=T[i];number++;
11.}
12. return(C)

```

TYPE B. Similar to the previous problem. Output can be longer though...

Problem 7. (30 POINTS)

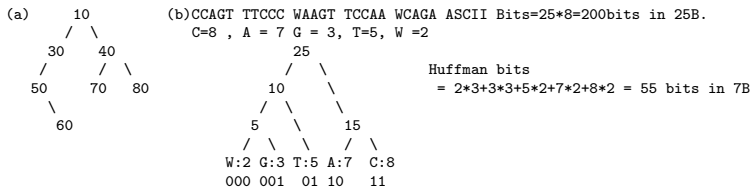
TYPE A. Stay away from MergeSort; it is not space efficient. Note that key q is fixed. Finding $x, y \in X, Y$ such that $x + y - q = 0$ is equivalent to finding a $x \in X$ such that $x = q - y$. We go through all Y elements and for each $y \in Y$ we binary search $q - y$ into X to determine if there is an x such that $x = q - y$ i.e. $x + y - q = 0$! We precondition X for binary search by sorting it in step 0. The running time of the algorithm is that of line 0, i.e. $O(n \lg n)$ plus the for loop's time. In each of the n iterations a binary search of $O(\lg n)$ is performed. Thus the cumulative running time is still $O(n \lg n)$. HeapSort is space-efficient and preferable to merge-sort as you need to 'Use as little extra space as possible'.

TYPE B. Stay away from MergeSort; it is not space efficient. Note that key q is fixed. Finding $a, b \in A, B$ such that $a - b - q = 0$ is equivalent to finding a $a \in A$ such that $a = q + b$. We go through all B elements and for each $b \in B$ we binary search $b + q$ into A to determine if there is an a such that $a = b + q$ i.e. $a - q - b = 0$! We precondition A for binary search by sorting it in step 0. The running time of the algorithm is that of line 0, i.e. $O(n \lg n)$ plus the for loop's time. In each of the n iterations a binary search of $O(\lg n)$ is performed. Thus the cumulative running time is still $O(n \lg n)$. HeapSort is space-efficient and preferable to merge-sort as you need to 'Use as little extra space as possible'.

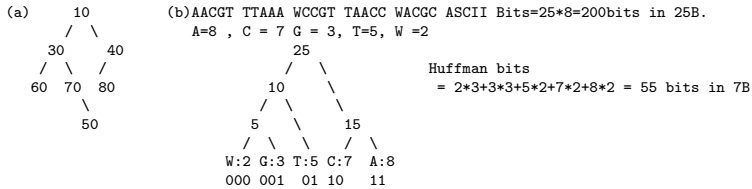
<pre> TYPEA(X,Y,n,q) 0. HeapSort(X); // In-place and time-efficient 1. for(i=0;i<n;i++){ 2. y = Y[i]; 3. key = q-y; 4. if (index = BinSearch(key,X))!= -1 { 5. x = X[index]; 6. return(x,y); // x=q-y i.e. x+y-q=0 7. } 8. return(NOTFOUND); </pre>	<pre> TYPEB(A,B,m,q) 0. HeapSort(A); 1. for(i=0;i<m;i++){ 2. b = B[i]; 3. key = b+q; 4. if (index = BinSearch(key,A))!= -1 { 5. a = A[index]; 6. return(a,b) // a = b +q i.e. a-b-q=0 7. } 8. return(NOT FOUND); </pre>
---	---

Problem 8. (60 POINTS)

TYPE A



TYPE B



Problem 9. (30 POINTS)

TYPE A and B. Build a MinHeap on the n or m keys as needed using no more than $2n$ and $2m$ comparisons, i.e. in time $O(n)$ and $O(m)$ respectively. This is time and space efficient and also comparison efficient. The perform 10 ExtracMin operations for TYPE A, and 8 ExtractMin operations for TYPE B, and fill the output left to right 1st smaller (min) to last (10th or 8th) smaller in sorted order. This would take $O(\lg n)$ or $O(\lg m)$ time or comparisons per key or for all 10 or 8 keys, and be done space efficiently as well (constant space used other than the Heap array). Total time is $O(n)$ and $O(m)$ respectively and other than the array used to build the binary heap a constant number of registers has been used. The comparison count is $2n + O(\lg n)$ i.e. $2n + o(n)$ or $2m + O(\lg m)$ i.e. $2m + o(m)$. (The asymptotic notation hides an 8 or 10 as needed.)