CSI 3105 Assignment 2 , 2014
Solutions

1.a) Worst case analysis (assuming n is a power of 2, i.e. $n = 2^k$):

Input size: n, the number of integers in X
Basic operations counted: Additions involving X and using the max function
Input of size n which gives worst case: All cases the same

Analysis:
If n > 1: In this case we have two for loops, each with two basic operations per iteration, and each for loop executes n/2 times, for a total of 2n basic operations. We also have two other basic operations to count (at the line Maxcrossing ..., and the last line return...) which occur outside the for loops, plus the basic operations that occur inside the two recursive calls to the algorithm MaxSum. Each recursive call to MaxSum involves looking at a range of n/2 numbers in X, and so each requires W(n/2) basic operations. So in total we have:

$W(n) = 2n + 2 + 2W(n/2)$.

If n = 1: In this case (which is the base case) we have just one max operation, and so W(1) =1.

So the recurrence relation for W(n) is:
$W(n) = 2W(n/2) + 2n + 2$ for $n > 1$
$W(1) = 1$.

(Note: The algorithm never reaches the case n = 0 recursively, and 0 is not a power of 2, hence we do not consider n=0)

Solving the recurrence relation using back substitution (recall $n = 2^k$):

$W(2^k) = 2W(2^{k-1}) + 2*2^k + 2$
$= 2(2W(2^{k-2}) + 2*2^{k-1} + 2) + 2*2^k + 2$
(since $W(2^{k-1}) = 2W(2^{k-2}) + 2*2^{k-1} + 2$ by the recurrence relation)

$= 2^2*W(2^{k-2}) + 2*2^{k+1} + 2^2 + 2$ (simplifying and regrouping)
.
. (note that you could do one more substitution here, if you weren't sure of the pattern yet)
.
$= 2^i *W(2^{k-i}) + i*2^{k+1} + 2^i + 2^{i-1} + ... + 2$
.

.
.

$= 2^k * W(2^{k-k}) + k*2^{k+1} + 2^k + 2^{k-1} + ... + 2$ (when i=k)

$= 2^k + k*2^{k+1} + \Sigma(2^j$: for j = 1 to k) (using W(1) = 1)

$= 2^k + k*2^{k+1} + (\ (2^{k+1} - 1) - 2^0\ )$ (using the formula for the summation, Appendix A. Note that we must subtract 2^0 here, since the formula starts with j=0)

$= n + (\lg n)*2n + 2n - 1 - 1$ (using $n=2^k$, $k = \lg n$)

$= 2n\lg n + 3n - 2$ (simplifying)

2. We will use the log relation $\lg n = (\lg e)* (\ln n)$ .....(*)
and L'Hôpital's Rule .....(LH)

a) Let $g(n) = \lg n$, let $f(n) = \text{sqrt}(n) = n^{1/2}$.

$$\lim_{n\to\infty} \frac{g(n)}{f(n)}$$

$$= \lim_{n\to\infty} \frac{\lg n}{\text{sqrt}(n)}$$

$$= \lim_{n\to\infty} \frac{(\lg e)(\ln n)}{n^{1/2}}$$ .....using (*)

$$= \lim_{n\to\infty} \frac{(\lg e)/n}{\frac{1}{2} n^{-1/2}}$$ .....using (LH)

$$= \lim_{n\to\infty} \frac{(\lg e)}{\frac{1}{2} n^{1/2}}.$$ .....simplifying

$$= \lim_{n\to\infty} \frac{0}{\frac{1}{4} n^{-1/2}}$$ .....using LH

$= 0$. So we have g(n) is O(f(n)) as required.

b) Let $g(n) = 20n^2 + 20n + 7$, let $f(n) = n + 5$.

$$\lim_{n\to\infty} \frac{g(n)}{f(n)}$$

$$= \lim_{n\to\infty} \frac{20n^2 + 20n + 7}{n + 5}$$

$$= \lim_{n\to\infty} \frac{40n + 20}{1} \quad \text{.....using (LH)}$$

$= \infty$. Thus $g(n)$ is $\Omega(f(n))$ as required.

c) Let $c = 15{,}000$ and $n_o = 1$. Then $n^3 + 5000n^2 + 20 \leq 5000n^3 + 5000n^3 + 5000n^3$
$\leq 15{,}000n^3$ for all $n \geq 1$.
So $n^3 + 5000n^2 + 20 \leq cn^3$ for all $n \geq n_o$, as required.

3.
Input Size: 3, number of keys in S.
Basic operation counted: number of comparisons of keys.
Analysis:
This analysis is under the assumption that the keys are distinct. (If we wanted to also do the analysis for duplicate keys, we would simply need to include more input partitions).

Let S be [A, B, C].

Input type 1: $A < B < C$ $t(I_1) = 2$
(We split into A, BC, then BC into B, C: One comparison of B and C to merge B and C, then one comparison of A and B to merge A and BC)
Input type 2: $A < C < B$ $t(I_2) = 2$
Input type 3: $B < A < C$ $t(I_3) = 3$
Input type 4: $B < C < A$ $t(I_4) = 3$
Input type 5: $C < A < B$ $t(I_5) = 3$
Input type 6: $C < B < A$ $t(I_6) = 3$

$p(Ii) = 1/6$ as all input types are equally likely.

So $A(3) = 1/6(2 + 2 + 3 + 3 + 3 + 3) = 1/6(16) = 8/3$.

4.a) For the usual algorithm (improved): $W(n) = 2n^3 - n^2$ (it is $n^3 - n^2$ additions and n^3 multiplications). So for n=32: $W(n) = 64{,}512$.
For Strassen's algorithm: $W(n) = 7^{\lg n} + 6(7^{\lg n}) - 6n^2$ (it is $7^{\lg n}$ multiplications and $6(7^{\lg n}) - 6n^2$ additions and subtractions). So for n=32: $W(n) = 111{,}505$
So the usual algorithm(improved) is faster.

b) For n>4, W(n) = 7W(n/2)
For n=2, W(n) = 8
For n=1, W(n)=1.

5.
a)
Back substitution with recurrence relation:

$W(2^k) = 2W(2^{(k-1)}) + 4$ (using relation)
$= 2(2W(2^{(k-2)}) + 4)) + 4$ (using relation)
$= 2^2W(2^{(k-2)}) + 2*4 + 4$ (simplifying)
:
$= 2^iW(2^{(k-i)}) + 4(2^{i-1} + 2^{i-2} + ... + 1)$ (for the ith line, guessing the pattern)
:
$= 2^{(k-1)}W(2^{(k-(k-1))}) + 4(2^{k-2} + 2^{k-3} + ... + 1)$ (for (k-1) line, where we get to the base case: i.e. when $2^{(k-i)} = 2$)
$= 2^{k-1} + 4\Sigma(2^j: j = 0 \text{ to } k-2)$ (since W(2) = 1)
$= 2^{k-1} + 4(2^{k-1} - 1)$ (using formula for summation, in text)
$= 5(2^{k-1}) - 4$ (simplifying)

We now must get our formula in terms of n:
$W(n) = 5n/2 - 4$ (using $n/2 = 2^{k-1}$)

b) Prove our result using induction:

Claim: $W(n) = 5n/2 - 4$ for $n \geq 2$, n a power of 2. (#)
Proof of Claim:
Base Case: For n = 2, we have our recurrence relation gives W(2) = 1, and our claim formula (#) gives 5(2/2) - 4 = 1. They match!

Induction Hypothesis (IH)
The claim is true for all m such that $2 \leq m < n$, and m a power of 2,
i.e. $W(m) = 5m/2 - 4$ for all m such that $2 \leq m < n$, and m a power of 2.

Induction Step
Prove the claim is true for n, where $n \geq 3$, n a power of 2 (note that we have already shown it is true for n = 2).

$W(n) = 2(W(n/2)) + 4$ (by the recurrence relation)
$= 2(5(n/2)/2 - 4) + 4$ (since n/2 is a power of 2 and >=2 and <n, we can apply the IH to W(n/2))
$= 5n/2 - 8 + 4$ (simplifying)
$= 5n/2 - 4.$
Therefore, $W(n) = 5n/2 - 4$, as required.
c) $\Theta(n)$.

6.
Idea of algorithm: We start by looking in S in the index range 1 to n (i.e. all of S). We multiply the first and last number in the range, i.e. let R = X[1] * X[n].

Now, since the numbers are sorted and distinct and >0, we must have that R > X[1] * X[i] for any i != n (since X[n] > X[i]). So, if P is greater than R, then that means that P is greater than X[1] * (any number in the list). So if two of the numbers multiply together to get P, X[1] can't be one of them. In this case, the answer for all of S will be the same as the answer we get if we consider just the numbers indexed from 2,...,n. Similarly,

Similarly, since the numbers are sorted and distinct, we must have that R < X[i] * X[n] for any i!=1. So, if P is less than R, then that means that P is less than X[n] * (any number in the list). So if two of the numbers multiply together to get P, X[n] can't be one of them. In this case, the answer for all of S will be the same as the answer we get if we consider just the numbers indexed from 1,...,n-1.

What is the base case? When the range we are considering only has one number in it, the answer will clearly be false.

Pseudocode:

Input: array S of n positive distinct integers, integer P, and index range indicators low...high, where low <= high. We will assume that the array S, n and P are constants.
Output: a boolean which is true if there are two numbers in S that have product P, and false otherwise.

```
boolean MultPair(int low, int high, int P, int S[])
{
  int R;
  if (low == high)
    return false;
  else {
    R=S[low]*S[high];
    if (R == P)
      return true;
    if (R > P)
      return MultPair(low + 1, high);
    else
      return MultPair(low, high - 1);
  }
}
```