General Instructions

1. The exam contains four questions.

2. You may use the class notes and homework assignments. No books or any other printed/copied material is allowed. Electronic devices must be turned off.

3. The work is to be your own and you are expected to adhere to the UMass Boston honor system.

4. Write your answers in the available spaces, using the back of the page if needed. Write clearly and concisely and try to avoid cursive.

5. Please explain your answers if needed but do it briefly.

6. You may use any proof technique we showed in class or any other technique, as long as it constitutes a mathematical proof. Remember that a proof by example is generally good only to show that something is NOT true.

7. If you base your answer on a homework question state exactly which question it was.

Good Luck!

Name: __________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
1. (25%) Order of growth

Use the substitution method to show that $T(n) = 16T(n^\frac{2}{4}) + n^2 = O(n^2 \log n)$.
Assume $T(2) = d$ (some constant). I know this can be shown easily with the master theorem, but please use induction and do not skip stages.
2. Sorting (25%) In class we assumed that our input array for sorting contains \( n \) distinct elements but in general it does not have to be the case, and there may be elements with the same key. A sorting algorithm is \textit{stable} if elements with the \textit{same key} appear in the sorted array in the same order as they do in the input array. Show that both HeapSort and QuickSort are not stable by showing an example where two equal key elements change order during sorting. The pseudo-code for partition and heapsort are given for your convenience. \textbf{Hint:} For Quicksort, use the example in the class notes and make a small change.

\begin{algorithm}
\textbf{Algorithm 1} Partition(A,p,r) \\
1: \( x \leftarrow A[r] \) \( \triangleright \) \( x \) is the “pivot”. \\
2: \( i \leftarrow p - 1 \) \( \triangleright \) \( i \) maintains the “left-right boundary”. \\
3: \textbf{for} \( j \leftarrow p \) to \( r - 1 \) \textbf{do} \\
4: \quad \textbf{if} \( A[j] \leq x \) \textbf{then} \\
5: \quad \quad \( i \leftarrow i + 1 \) \\
6: \quad \quad \text{exchange} \ A[i] \leftrightarrow A[j] \\
7: \quad \textbf{end if} \\
8: \textbf{end for} \\
9: \text{exchange} \ A[i + 1] \leftrightarrow A[r] \\
10: \textbf{return} \ i + 1
\end{algorithm}

\begin{algorithm}
\textbf{Algorithm 2} Heapsort(A) \\
1: \textit{BuildHeap}(A) \\
2: \textbf{for} \( i \leftarrow \text{length}[A] \) to 2 \textbf{do} \\
3: \quad \text{exchange} \ A[1] \leftrightarrow A[i] \\
4: \quad \textit{heapsize}[A] \leftarrow \textit{heapsize}[A] - 1 \\
5: \quad \textit{Heapify}(A,1) \\
6: \textbf{end for}
\end{algorithm}
3. Sorting (25%) Give a simple $O(n)$ algorithm to rearrange an array of $n$ keys so that all the negative keys precede all the nonnegative keys. Your algorithm must be in-place, meaning you cannot allocate another array to temporarily hold the items. Explain briefly why it is $O(n)$
4. Heaps (25%): A min-heap is a heap such that the root is the smallest key in the heap and each one of its two children is a min-heap. Can a sorted array be a min-heap as-is? That is, without any rearrangements? Explain.