Problem 1. 17.1-1 p409
Consider a sequence of \( n \) PUSH, POP, MULTIPUSH and MULTIPOP operations on an initially empty stack. The worst-case cost sequence is a "large" MULTIPUSH followed by a "large" MULTIPOP, followed by another "large" MULTIPUSH, etc. If a large MULTIPUSH is size \( \Theta(n) \), then the cost of the MULTIPUSH is \( \Theta(n) \), and the cost of the MULTIPOP is size \( \Theta(n) \). The cost of this sequence of \( n \) operations is \( \Theta(n^2) \). Thus the \( O(1) \) bound on the amortized cost of stack operations does not hold.

Problem 2. 17.1-2 p409
Consider a \( k \)-bit binary counter with INCREMENT and DECREMENT. The worst-case cost sequence of \( n \) INCREMENT and DECREMENT operations is when an INCREMENT flips \( k \) bits, followed by alternating DECREMENT and INCREMENT operations. Each of the \( n \) operations will cost \( \Theta(k) \), for a total of \( \Theta(nk) \) time.

Assume \( k = 3 \). Then alternating the following takes 3 flips each: add 1 to 111 which results in 000. Subtracting 1 from 000 which results in 111.

Problem 3. 17.1-3 p410
Let \( c_i = \text{cost of the } i\text{th operation} \).

\[
c_i = \begin{cases} 
  i & \text{if } i \text{ is an exact power of 2}, \\
  1 & \text{otherwise}.
\end{cases}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
\( n \) operations cost:
\[
\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \\
\leq n + n + \frac{n}{2} + \frac{n}{4} + \ldots + 2 + 1 \\
\leq n + \frac{n - \frac{1}{2}}{1 - \frac{1}{2}} \\
= n + (2n - 1) \\
< 3n
\]

By aggregate analysis, the amortized cost per operation is \( O(1) \).

**Problem 4. 18.2-1 p447**

attached

**Problem 5. 19.2-10 p473**

Represent \( n \) in binary, i.e. \( n = (n_k, \ldots, n_0) \), where \( k = \lfloor \log n \rfloor \). A binomial heap of size \( n \) has one tree per one in the binary representation. \( n_k \) is always one and \( 2^k = \Omega(\log n) \). Let \( \text{ones}(n) \) be the number of ones in the binary representation for \( n \). When \( n \) is a power of two, then \( \text{ones}(n) = 1 \). However when \( n \) is one less than a power of two, then \( \text{ones}(n) = k + 1 = \lfloor \log n \rfloor + 1 \).

a) \( \Omega(\log n) \)

- **BINOMIAL-HEAP-EXTRACT-MIN**: For any \( n \), let \( k \) be as defined above. Assume the minimum is the root of \( B_k \) (the biggest tree). We remove this tree from the heap, take off the root of it, reverse the order of the \( k \) children and make a heap out to this list. Finally we and merge this heap with the remaining heap. This takes \( \Omega(k) = \Omega(\log n) \).

- **BINOMIAL-HEAP-DECREASE-KEY**: For any \( n \), let \( k \) be as defined above. Assume that the key to be decreased is one of the highest depth leaves in \( B_k \). So decrease can take \( \Omega(k) = \Omega(\log n) \) in the worst case.

- **BINOMIAL-HEAP-DELETE**: Since this uses both **BINOMIAL-HEAP-DECREASE-KEY** and **BINOMIAL-HEAP-EXTRACT-MIN** as subroutines, we can use either of the above examples to get the lower bound \( \Omega(\log n) \).

b) \( \Omega^\infty(\log n) \)

- **BINOMIAL-HEAP-INSERT**: Whenever \( n \) is a power of two, insert is \( O(1) \). So for any \( n_0 \) there is always an \( n \) larger than \( n_0 \) for which insert is \( O(1) \). This means that this operation is not \( \Omega(\log n) \). However whenever \( n \) is one less than a power of two, then increment is \( \Omega(k) \), where \( k \) is as defined above. This is because in that case there are exactly \( k + 1 \) trees and all \( k + 1 \) trees are merged into one tree during the insert: \( 1^{k+1} + 1 = 10^{k+1} \). Recall that \( k = \lfloor \log n \rfloor \) and there are infinitely many powers of two. Therefore this operation is \( \Omega^\infty(\log n) \).

- **BINOMIAL-HEAP-MINIMUM**: When \( n \) is a power of two then this op. is again \( O(1) \). So it can’t be \( \Omega(\log n) \) by the same reasoning as above. Similarly, when \( n \) is one less than a power of two then \( k + 1 \) roots have to be checked costing order \( k \) work. Therefore this operation is \( \Omega^\infty(\log n) \).

- **BINOMIAL-HEAP-UNION**: Note that here \( n \) specifies the total size of both heaps.

This op. is \( \Omega^\infty(\log n) \) because item **BINOMIAL-HEAP-INSERT** is a special case: Let \( n \) be a power of two and insert one more element into a heap of size \( n - 1 \). ...
We can also show that the BINOMIAL-HEAP-UNION is $\Omega(\log n)$ (this op. should have been part of the a) list): For any $n$, let $k$ be as above. Merging two heaps of size $2^k - 1$ and $n - 2^k + 1$ involves reading the root key of at least $k$ trees and thus costs $\Omega(k) = \Omega(\log n)$.

**Problem 6. 28.2-4 p741**

\[
\begin{align*}
\log_{68} 132464 &= 2.79513 \\
\log_{70} 143640 &= 2.79512 \\
\log_{72} 155424 &= 2.79515
\end{align*}
\]

This shows that the method of multiplying 70x70 matrices with 143640 multiplications yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm. The running time is $O(n^{2.80})$, and this beats Strassen’s running time of $O(n^{2.81})$.

**Problem 7. 28.2-6 p741**

Multiply the complex numbers $a + bi$ and $c + di$ using only 3 real multiplications. The algorithm should take $a, b, c,$ and $d$ as input and produce the real component $ac - bd$ and the imaginary component $ad + bc$ separately.

\[
\begin{align*}
M_1 &= (a - b)(c + d) = ac + ad - bc - bd \\
M_2 &= bc \\
M_3 &= ad
\end{align*}
\]

Then

\[
\begin{align*}
ac - bd &= M_1 + M_2 - M_3 \\
ad + bc &= M_3 + M_2
\end{align*}
\]