CHAPTER 2 – PROBLEMS

Problem 1. Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1, \ldots, n-1$.

Let's take the left child of the node indexed by $\left\lfloor \frac{n}{2} \right\rfloor$

$$\text{LEFT}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) = 2\left\lfloor \frac{n}{2} \right\rfloor \geq 2\frac{n}{2} = n$$

Since the index of the left child is larger than the index of the last element in the heap ($n-1$), this node doesn't have children and thus is a leaf. Same goes for all nodes with larger indices.

Problem 2. How would you modify Quicksort to sort an array of integers by decreasing order.

Below is just one possible implementation:

```
void QuickSort(int arr[], int low, int high)
{
	if (low >= high)
		return;
	int p = PartitionFirst(arr, low, high);
	QuickSort(arr, low, p - 1);
	QuickSort(arr, p + 1, high);
}

//partition so that the pivot is the first element
int PartitionFirst(int arr[], int start, int end)
{
	int p = start; //pivot is the first element
	int j = start + 1;
	for (int i = start + 1; i <= end; i++)
	{
		if (arr[i] < arr[p])
			continue;
		if (i > j)
			Swap(arr[i], arr[j]);
		j++;
	}
	Swap(arr[p], arr[j-1]);
	return j-1;
}
```

Problem 3. What is the running time of Quicksort when all elements of array A have the same value?

→ It is the worst case of Quicksort in which 1 sub array of partition have $n-1$ elements and one has no element. Therefore, the running time of Quicksort in this case is n^2.

Problem 4. What is the running time of heapsort on an array A of length n that is already sorted in increasing order? What about decreasing order?

- If the array is sorted in increasing order, the algorithm will need to convert it to a heep that will take O(n). Afterwards, however, there are n−1 calls to MAX-HEAPIFY and each one will perform the full $\log_2 k$ operations. Since:

$$\sum_{i=1}^{n} \log_2 k = \log_2((n-1)!) = O(n \log_2 n)$$

- Same goes for decreasing order. BUILD-MAX-HEAP will be faster (by a constant factor), but the computation time will be dominated by the loop in HEAPSORT, which is $O(n \log_2 n)$

Problem 5. Show how to sort n integers in the range 0 to $n^2 - 1$ in O(n) time.

The idea is to use Radix sort:

```
for i←0 to d-1 do
    use Counting sort to sort array A on digit i
```

→ The running time is $O(d(n+k))$ where k is the possible values of each digit of the integers in A. For decimal system, $k = 10$.

Since max value of each integer is $n^2 - 1$, the value of d would be $O(\log_k n)$. Hence, the running time would be $O(\log_k n\,(n+k))$. If we choose $k = n$, then running time is $O\left(\log_n n\,(n+k)\right) = O(n)$.