Section 09: Solutions

1. Sorting Algorithms Steps

Show the steps taken for each sort as we have learned in lecture.

(a) Insertion sort on `0, 4, 2, 7, 6, 1, 3, 5`.

Solution:

```
0 | 4 2 7 6 1 3 5
0 4 | 2 7 6 1 3 5
0 2 4 | 7 6 1 3 5
0 2 4 7 | 6 1 3 5
0 2 4 6 7 | 1 3 5
0 1 2 4 6 7 | 3 5
0 1 2 3 4 6 7 | 5
0 1 2 3 4 5 6 7 |
```

(b) Selection sort on `0, 4, 2, 7, 6, 1, 3, 5`.

Solution:

```
0 | 4 2 7 6 1 3 5
0 1 | 2 7 6 4 3 5
0 1 2 | 7 6 4 3 5
0 1 2 3 | 6 4 7 5
0 1 2 3 4 | 6 7 5
0 1 2 3 4 5 | 7 6
0 1 2 3 4 5 6 | 7
0 1 2 3 4 5 6 7 |
```

(c) Heapsort on `0, 6, 2, 7, 4`. (You may want to draw out the heap. Make sure the first step you do is the heapification step!)

Solution:

```
7 6 2 0 4 (turns the array into a valid heap)
6 4 2 0 7 ('delete' 7, then sink 4)
4 0 2 6 7 ('delete' 6, then sink 0)
2 0 4 6 7 ('delete' 4, then sink 2)
0 2 4 6 7 ('delete' 2)
0 2 4 6 7 ('delete' 0)
```

(d) Merge sort on `0, 4, 2, 7, 6, 1, 3, 5`.

Solution:

$\boxed{0\,4\,2\,7\,6\,1\,3\,5}$

$\boxed{0\,4\,2\,7}\ \boxed{6\,1\,3\,5}$

$\boxed{0\,4}\ \boxed{2\,7}\ \boxed{6\,1}\ \boxed{3\,5}$

$\boxed{0}\ \boxed{4}\ \boxed{2}\ \boxed{7}\ \boxed{6}\ \boxed{1}\ \boxed{3}\ \boxed{5}$

$\boxed{0\ 4}\ \boxed{2\ 7}\ \boxed{1\ 6}\ \boxed{3\ 5}$

$\boxed{0\ 2\ 4\ 7}\ \boxed{1\ 3\ 5\ 6}$

$\boxed{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}$

(e) Quicksort on `18, 7, 22, 34, 99, 18, 11, 4`. (Assume that we always choose first element as the pivot. Show the steps taken at each partitioning step.)

Solution:

Partition(18):
`7, 11, 4, 18, 22, 34, 99, 18`

Partition(7):
`4, 7, 11, 18, 22, 34, 99, 18`

Partition(4): (no change, sublist containing 4 has one element)
`4, 7, 11, 18, 22, 34, 99, 18`

Partition(11): (no change, sublist containing 11 has one element)
`4, 7, 11, 18, 22, 34, 99, 18`

Partition(22):
`4, 7, 11, 18, 18, 22, 34, 99`

Partition(18): (no change, sublist containing 18 has one element)
`4, 7, 11, 18, 18, 22, 34, 99`

Partition(34):
`4, 7, 11, 18, 18, 22, 34, 99`

Partition(99): (no change, sublist containing 99 has one element)
`4, 7, 11, 18, 18, 22, 34, 99`

2. Sorting Decisions

For each of the following scenarios, say which sorting algorithm you think you would use and why. As with the design decision problems, there may be more than one right answer.

(a) Suppose we have an array where we expect the majority of elements to be sorted "almost in order". What would be a good sorting algorithm to use?

Solution:

Merge sort and quick sort are always predictable standbys, but we may be able to get better results if we try using something like insertion sort, which is $\mathcal{O}(n)$ in the best case.

(b) You are writing code to run on the next Mars rover to sort the data gathered each night (Think about sorting with limited memory and computational power).

Solution:

Since each memory stick costs thousands (millions?) of dollars to send to Mars, an in-place sort is probably your best bet. Among in-place sorts, heap sort is a great choice (since it is guaranteed $\mathcal{O}(n \log n)$ time and doesn't even use much stack memory). Insertion sort meets memory needs, but wouldn't be fast.

(c) You're writing the backend for the website `SortMyNumbers.com`, which sorts numbers given by users.

Solution:

Do you trust your users? I wouldn't. Because of that, I want a worst-case $\mathcal{O}(n \log n)$ sort. Heap sort or Merge sort would be good choices.

(d) Your artist friend says for a piece she wants to make a computer sort every possible ordering of the numbers $1, 2, \dots, 15$. Your friend says something special will happen after the last ordering is sorted, and you'd like to see that ASAP.

Solution:

Since you're going to sort all the possible lists, you want to optimize for the average case – Quick sort has the best average case behavior, which makes it a really good choice. Merge sort and heapsort also have average speed of $\mathcal{O}(n \log n)$ but they're usually a little slower on average (depending on the exact implementation).

She didn't appreciate your snarky suggestion to "just print $[1, 2, \dots, 15]$ $15!$ times." Something about not accurately representing the human struggle.

3. Sorting Algorithm Analysis

(a) What are two techniques that can be used to reduce the probability of Quicksort taking the worst case running time?

Solution:

(i) Randomly choose pivots.

(ii) Shuffle the list before running Quicksort.

(b) When choosing an appropriate algorithm, there are often several trade-offs that we need to consider. Complete the chart for the following sorting algorithms by writing down the best case and worst case runtimes in $\Theta()$ notation and whether or not the sort is stable. You may write any notes about the sort in the "Notes" column; this column will not be graded.

	Runtime (best)	Runtime (worst)	Stable? (Y/N)	Notes (not graded)
Selection Sort				
Insertion Sort				
Heapsort				
Merge Sort				
Quicksort (Naive)				

Solution:

	Runtime (best)	Runtime (worst)	Stable? (Y/N)	Notes (not graded)
Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	No	Same best case and worst case. Not stable if we use swapping method from lab, but there is a possibility of stability if we use an auxiliary array .
Insertion Sort	$\Theta(N)$	$\Theta(N^2)$	Yes	Best case is using a sorted array or an array of all duplicates. Worst case is using a reverse-sorted array. "Average" case is $\Theta(N^2)$. The runtime of insertion sort depends on the number of inversions; if we know the number of inversions is k, then the runtime of insertion sort is $\Theta(N+k)$ (the best case is when k is the smallest and is equal to 0 (sorted array or array of duplicates); the worst case is when k is the largest and is equal to $\frac{(N(N-1)}{2}$ (maximum number of inversions, reverse sorted array)).
Heapsort	$\Theta(N)$	$\Theta(N \log N)$	No	Best case is if the heap contains all duplicate items (every remove operation wouldn't require bubbling the heap).
Merge Sort	$\Theta(N \log N)$	$\Theta(N \log N)$	Yes	
Quicksort (Naive)	$\Theta(N \log N)$	$\Theta(N^2)$	Yes	Quicksort can also be implemented so it's unstable if Hoare's partitioning is used (no auxiliary arrays are created, and it is done "in-place").