Noel Ball
HW5

7.1-1.) Partition on array (italic i, underline j, bold r)

< | 13 19 9 5 12 8 7 4 21 2 6 | **11** >
< | 13 19 9 5 12 8 7 4 21 2 6 | **11** >
< 13 | 19 9 5 12 8 7 4 21 2 6 | **11** >
< 13 19 | 9 5 12 8 7 4 21 2 6 | **11** >
< *9* | 19 13 | 5 12 8 7 4 21 2 6 | **11** >
< *9 5* | 13 19 | 12 8 7 4 21 2 6 | **11** >
< *9 5* | 13 19 12 | 8 7 4 21 2 6 | **11** >
< *9 5 8* | 19 12 13 | 7 4 21 2 6 | **11** >
< *9 5 8 7* | 12 13 19 | 4 21 2 6 | **11** >
< *9 5 8 7 4* | 13 19 12 | 21 2 6 | **11** >
< *9 5 8 7 4* | 13 19 12 21 | 2 6 | **11** >
< *9 5 8 7 4 2* | 19 12 21 13 | 6 | **11** >
< *9 5 8 7 4 2 6* | 12 21 13 19 | **11** >
< *9 5 8 7 4 2 6* | **11** | 21 13 19 12 >

7.1-3.) Partitions two operations comparing and swapping elements both take $\Theta(n)$ time, resulting in $\Theta(n)$ for partition on a subarray.

7.2-1.) Assuming $T(n) = cn^2$ for some constant c. The base case $T(1) = c*1^2 = c$ shows any non-negative c is a valid constant. Since $T(n-1) = c(n-1)^2$, then $T(n) = c(n-1)^2 + \Theta(n)$. Expanding the exponent, $T(n) = c(n^2 - 2n + 1) + \Theta(n)$. Given the order of growth of the term, $T(n) = \Theta(n^2)$

7.2-2.) The running time of quicksort when every element is the same value is $\Theta(n^2)$ because every partition will result as unbalanced with only one side to the pivot, causing the recursive call to sort every element with recurrence $T(n) = T(n-1) + \Theta(n)$.

7.3-1.) Random algorithms are discussed in terms of expected runtime because the worst case is typically such a low probability that the expected running time better reflects the algorithm's average running time in practice.

7.4-2.) The best case of quicksort takes $\Theta(n)$ to compare elements to the pivot in partition, as well as making lgn recursive calls due to the pivot splitting the array into two perfect halves, resulting in a $\Omega(n\lg n)$ runtime.

8.1-2.) $\lg(n!)$ can be expanded as $\lg(n(n-1)(n-2) \ldots (2)(1))$ and separated $\lg(n) + \lg(n-1) + \ldots + \lg(1)$ showing $n\lg(n) \geq \lg(n!) \geq \frac{n}{2}\lg(\frac{n}{2})$, since it is within $O(n\lg n)$ and $\Omega(n\lg n)$, $\lg(n!) = \Theta(n\lg n)$

8.2-1.) Counting sort on array

A < 6 0 2 0 1 3 4 6 1 3 2 > , C < 2 2 2 2 1 0 2 >
C < 2 4 6 8 9 9 11 >
B < _ _ _ _ _ 2 _ _ _ _ _ > , C < 2 4 5 8 9 9 11 >
B < _ _ _ _ _ 2 _ 3 _ _ _ > , C < 2 4 5 7 9 9 11 >
B < _ _ _ 1 _ 2 _ 3 _ _ _ > , C < 2 3 5 7 9 9 11 >
B < _ _ _ 1 _ 2 _ 3 _ _ 6 > , C < 2 3 5 7 9 9 10 >
B < _ _ _ 1 _ 2 _ 3 4 _ 6 > , C < 2 3 5 7 8 9 10 >
B < _ _ _ 1 _ 2 3 3 4 _ 6 > , C < 2 3 5 6 8 9 10 >
B < _ _ 1 1 _ 2 3 3 4 _ 6 > , C < 2 2 5 6 8 9 10 >
B < _ 0 1 1 _ 2 3 3 4 _ 6 > , C < 1 2 5 6 8 9 10 >
B < _ 0 1 1 2 2 3 3 4 _ 6 > , C < 1 2 4 6 8 9 10 >
B < 0 0 1 1 2 2 3 3 4 _ 6 > , C < 0 2 4 6 8 9 10 >
B < 0 0 1 1 2 2 3 3 4 6 6 >

8.2-2.) Counting sort places elements of equal value in the output array in the same order they are input by working backwards through the input and putting the element at the furthest index available to that value due to the counting auxiliary array, with further elements of equal value being encountered further towards the beginning of the input at the next furthest index available, therefore it is stable.

8.3-1.) Radix-Sort on words

COW	SEA	TAB	BAR
DOG	TEA	BAR	BIG
SEA	MOB	EAR	BOX
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX	DIG	DIG	EAR
TAB	BIG	BIG	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX	FOX	RUG	TEA

8.4-1.) Bucket sort on array
A = < .79 .13 .16 .64 .39 .20 .89 .53 .71 .42 >
B = < (/) (.13 → .16/) (.20/) (.39/) (.42/) (.53/) (.64/) (.71 → .79/) (.89/) (/) >

8.4-3.) X can be 2 with HH, 1 with HT or TH, and 0 with TT, each with a $\frac{1}{4}$ probability, making $E[X] = \frac{2}{4} + \frac{1}{4} + \frac{1}{4} + \frac{0}{4} = 1$. That makes $E[X^2] = \frac{4}{4} + \frac{2}{4} + \frac{2}{4} + \frac{0}{4} = 2$, however $E^2[X] = 1^2 = 1$.