## **Quick Sort**

```
// Pre-condition: low and high are value indices into numbers.
// Post-condition: The values in numbers will be sorted in between
11
                indices low and high
void quicksort(int* numbers, int low, int high) {
   // Only have to sort if we are sorting more than one number
   if (low < high) {
      int split = partition(numbers,low,high);
      quicksort(______);
      quicksort(______
                                                             );
   }
}
// Swaps the values pointed to by a and b.
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

1-		-	-	-	-				-	
vais	Q	2	6	0	2	Λ	1	Δ	7	5
	0	3	U	9		4	L	U	/	3

If we call quicksort(vals, 0, 9) (assume 6 is the partition element) fill in split and what the following recursive calls would contain:

split =	
quicksort(	)
quicksort(	)

## **<u>Quick Sort</u>**

Assume the 1 <sup>st</sup> time partition is called, i = 2. Show the contents of vals after each iteration of the while loop After 1 <sup>st</sup> Loop: After 2 <sup>nd</sup> Loop: After 2 <sup>nd</sup> Loop: After 2 <sup>nd</sup> Loop: After 3 <sup>nd</sup> Loop: After 3 <sup>nd</sup> Loop: After and Loop	vals		8	3	6	9	2	4	1	0	7	5
After 1 <sup>st</sup> Loop:         After 2 <sup>nd</sup> Loop:         After 3 <sup>nd</sup> Loop:         After and the right spot:         After and the right spot:         After and the right and the value and the v	Assume	the	1 <sup>st</sup> time	partition	is called	i = 2. S	Show the	contents	of vals af	ter each	iteration	of the while lo
After 2 <sup>nd</sup> Loop: After 2 <sup>nd</sup> Loop: After 3 <sup>rd</sup> Loop: After and Loop: After putting partition in the right spot: // Returns the partition index such that all the values stored in vals from low // Returns the partition index such that all the values stored in vals from low // to partition are < partition & all the vals from partition to high are > . int partition(int* vals, int low, int high) { int i, lowpos; if (low == high) return low; // A base case that should never really occur // Pick a random partition element and swap it into index low. i = low + rand()%(high=low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low <= high) { // Move the low pointer until we find a value too large for this side. while () low++; // Move high until we find a value too small for this side. while () high; if (low < high) // Swap the two values that were on the wrong side. swap(&vals[lowpos], &vals(high]); // Swap partition into right spot.				1								
After 3 <sup>rd</sup> Loop: After 3 <sup>rd</sup> Loop: After putting partition in the right spot: // Returns the partition index such that all the values stored in vals from low // Returns the partition index such that all the values stored in vals from low // to partition are < partition & all the vals from partition to high are > . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occur // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low <= high) { // Move the low pointer until we find a value too large for this side. while () low++; // Move high until we find a value too small for this side. while () high; if (low < high) // Swap the two values that were on the wrong side. swap(&vals[low]); // Swap partition into right spot.	AIGII		<u></u>									
After 3 <sup>rd</sup> Loop: After 3 <sup>rd</sup> Loop: After putting partition in the right spot: // Returns the partition index such that all the values stored in vals from low // Returns the partition index such that all the values stored in vals from low // to partition are < partition & all the vals from partition to high are > . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occur // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low <= high) { // Move the low pointer until we find a value too large for this side. while () low++; // Move high until we find a value too small for this side. while () high; if (low < high) // Swap the two values that were on the wrong side. swap(&vals[low]); // Swap partition into right spot.		L										
After putting partition in the right spot: After putting partition in the right spot: // Returns the partition index such that all the values stored in vals from low // to partition are < partition & all the vals from partition to high are > . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occurs // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[1]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low <= high) { // Move the low pointer until we find a value too large for this side. while () low++; // Move high until we find a value too small for this side. while () high; if (low < high) // Swap the two values that were on the wrong side. swap(&vals[low], &vals[high]); // Swap partition into right spot.	After 2 <sup>nd</sup>	<sup>d</sup> Lo	op:				-	1	1	7		
After putting partition in the right spot: After putting partition in the right spot: // Returns the partition index such that all the values stored in vals from low // to partition are < partition & all the vals from partition to high are > . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occurs // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[1]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low <= high) { // Move the low pointer until we find a value too large for this side. while () low++; // Move high until we find a value too small for this side. while () high; if (low < high) // Swap the two values that were on the wrong side. swap(&vals[low], &vals[high]); // Swap partition into right spot.												
After putting partition in the right spot: After putting partition in the right spot: // Returns the partition index such that all the values stored in vals from low // to partition are < partition & all the vals from partition to high are > . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occurs // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[1]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low <= high) { // Move the low pointer until we find a value too large for this side. while () low++; // Move high until we find a value too small for this side. while () high; if (low < high) // Swap the two values that were on the wrong side. swap(&vals[low], &vals[high]); // Swap partition into right spot.		1										
<pre>// Returns the partition index such that all the values stored in vals from lot // to partition are &lt; partition &amp; all the vals from partition to high are &gt; . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occur // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low &lt;= high) { // Move the low pointer until we find a value too large for this side while () low++; // Move high until we find a value too small for this side. while () high; if (low &lt; high) // Swap the two values that were on the wrong side. swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>	After 3 <sup>rd</sup>	<sup>1</sup> Lo	<u>op:</u>									
<pre>// Returns the partition index such that all the values stored in vals from lot // to partition are &lt; partition &amp; all the vals from partition to high are &gt; . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occur // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low &lt;= high) { // Move the low pointer until we find a value too large for this side while () low++; // Move high until we find a value too small for this side. while () high; if (low &lt; high) // Swap the two values that were on the wrong side. swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>		L										
<pre>// Returns the partition index such that all the values stored in vals from lot // to partition are &lt; partition &amp; all the vals from partition to high are &gt; . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occur // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low &lt;= high) { // Move the low pointer until we find a value too large for this side while () low++; // Move high until we find a value too small for this side. while () high; if (low &lt; high) // Swap the two values that were on the wrong side. swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>	After nu	ttin	o nartiti	on in the	right spor	t•						
<pre>// to partition are &lt; partition &amp; all the vals from partition to high are &gt; . int partition(int* vals, int low, int high) { int temp; int i, lowpos; if (low == high) return low; // A base case that should never really occurs // Pick a random partition element and swap it into index low. i = low + rand()%(high-low+1); temp = vals[i]; vals[i] = vals[low]; vals[low] = temp; lowpos = low; // Store the index of the partition element. low++; // Update our low pointer. while (low &lt;= high) { // Move the low pointer until we find a value too large for this side while () low++; // Move high until we find a value too small for this side. while () high; if (low &lt; high) // Swap the two values that were on the wrong side. swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>	<u>ritter pu</u>		5 purities									
<pre>while (low &lt;= high) {     // Move the low pointer until we find a value too large for this sid     while () low++;     // Move high until we find a value too small for this side.     while () high;     if (low &lt; high) // Swap the two values that were on the wrong side.     swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>	// to j int pa in in if // i te va va	par <b>rti</b> t t (1 Pi mp ls[ .ls[	<pre>tition tion(i emp; , lowp .ow == .ck a p .ow + p = vals i] = v low] =</pre>	n are < int* va pos; high) random rand()% s[i]; vals[lo = temp;	partit <b>ls, int</b> return partiti (high-1 w];	low; con ele	all the int hig // A ba ement ar	e vals <b>jh) {</b> ase cas nd swap	from pa e that it int	should	n to h: I never x low.	igh are > .
<pre>// Move the low pointer until we find a value too large for this sid while () low++; // Move high until we find a value too small for this side. while () high; if (low &lt; high) // Swap the two values that were on the wrong side. swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>	1	Low	++;	// Up	date o	ur low	pointe	r.				
<pre>while () high; if (low &lt; high) // Swap the two values that were on the wrong side.     swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>	Ţv	whi	// N	love th	e low p						-	
<pre>swap(&amp;vals[low], &amp;vals[high]); } swap(&amp;vals[lowpos], &amp;vals[high]); // Swap partition into right spot.</pre>												
			5	swap(&v	als[low	/], &va	ls[high	n]);				
		-	-	_			_	_	_		_	nt spot.