2019A7PS0020U Dhruv Maheshwari

Lab 7

Ex 1:

Function	Big O

void swap (int *a, int *b)
{
int temp=*a;
*a=*b;
*b=temp;
}
	O(1)[declaration and assignment]

int partition(int array[], int low, int high)
{
int pivot= array[high];
int i=(low-1);
 	
for(int j=low; j<=high-1; j++)
{
if(array[j]<pivot)
{
i++;
swap(&array[i],&array[j]);
}
}
 	
swap(&array[i+1], &array[high]);
return (i+1);
}	

void quicksort(int array[], int low, int high)
{
if (low<high)
{
int x= partition(array, low, high);
quicksort(array, low, x-1);
quicksort(array, x+1, high);
}
}
	n
	/ \
	0 n-1
	/ \
	0 n-2
	
	cn + c(n-1) + c(n-2).....+2c
	$c(\frac{n(n+1)}{2} - 1)$
	$O(n^2)$

void merge(int array[], int l, int m, int r)
{
int i, j, k;
int n1=m-l+1;
int n2=r-m;
 	
int L[n1], R[n2];
 	
for (i = 0; i < n1; i++)
L[i] = array[l + i];
for (j = 0; j < n2; j++)
R[j] = array[m + 1 + j];	

```
   i = 0;
   j = 0;
   k = l;
   while (i < n1 && j < n2)
   {
      if (L[i] <= R[j])
      {
         array[k] = L[i];
         i++;
      }
      else
      {
         array[k] = R[j];
         j++;
      }
      k++;
   }

   while (i < n1)
   {
      array[k] = L[i];
      i++;
      k++;
   }

   while (j < n2)
   {
      array[k] = R[j];
      j++;
      k++;
   }
}
```

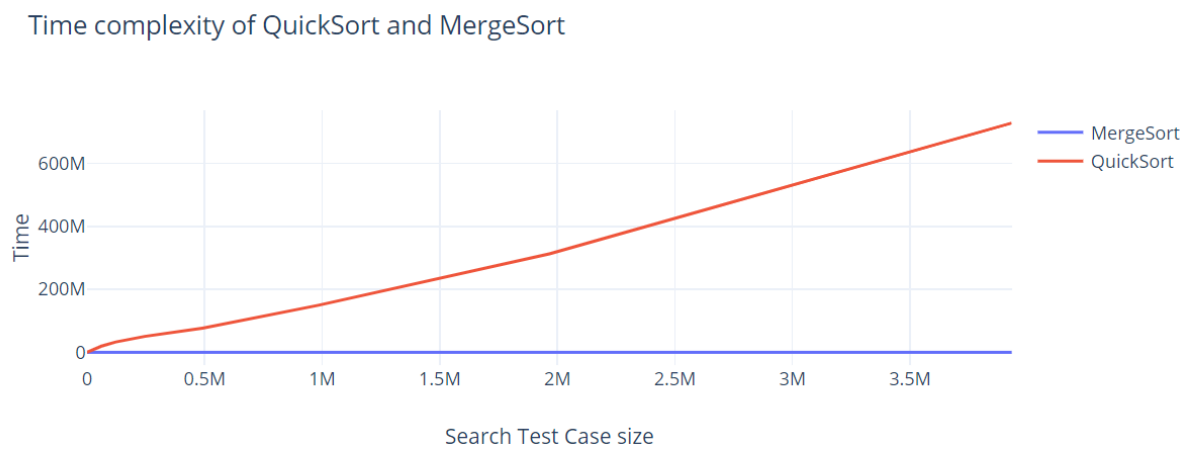
```
void mergeSort(int array[], int l, int r)
{
   if (l < r)
   {
      int m = l + (r - l) / 2;
      mergeSort(array, l, m);
      mergeSort(array, m + 1, r);

      merge(array, l, m, r);
   }
}
```

T(n) = $\begin{cases} b \ if \ n = 1 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + cn \end{cases}$

T(n) =2(2T(n/2^2) + cn/2) + cn
T(n) =2^2T(n/2^2) + 2cn
T(n) =2^2T(n/2^2) + 2cn
T(n) =2^kT(n/2^k) + kcn

n/2^k = 1 => k=ln n
T(n) = nb + cn*ln n
O(n)=n*ln n

2019A7PS0020U Dhruv Maheshwari



Ex 2:

Function	Big O

void Split(Node* source,Node** frontRef, Node** backRef)
{
Node* fast;
Node* slow;
slow = source;
fast = source->next;
 	
while (fast != NULL) {
fast = fast->next;
if (fast != NULL) {
slow = slow->next;
fast = fast->next;
}
}
 	
*frontRef = source;
*backRef = slow->next;
slow->next = NULL;
}
	While loop would continue for n-1 times where n is the number of elements in source node.
	
	Hence O(n)

Node* SortedMerge(Node* a, Node* b)
{
Node* result = NULL;
 	
if (a == NULL)
return (b);
else if (b == NULL)
return (a);
 	
if (a->data <= b->data) {
result = a;
result->next = SortedMerge(a->next, b);
}	

else {
result = b;
result->next = SortedMerge(a, b->next);
}
return (result);
}
	O(n)[as n comparisons]

void MergeSort(Node** headRef)
{
Node* head = *headRef;
Node* a;
Node* b;
 	
if ((head == NULL) || (head->next == NULL)) {
return;
}
 	
Split(head, &a, &b);
 	
MergeSort(&a);
MergeSort(&b);
 	
*headRef = SortedMerge(a, b);
}
	O(n)[discussed above]
	
	O(n)
	
	O(n)

void printList(Node* node)
{
while (node != NULL) {
cout << node->data << " ";
node = node->next;
}
}
	O(n)[elements in node]

void push(Node** head_ref, int new_data)
{
Node* new_node = new Node();
new_node->data = new_data;
new_node->next = (*head_ref);
(*head_ref) = new_node;
}
	O(1)[declaration, assignment]

```
Enter no of elements to be sorted: 6
Enter element: 0
Enter element: -5
Enter element: 8
Enter element: -10
Enter element: 23
Enter element: 4

UnSorted Linked List is:
4 23 -10 8 -5 0
Sorted Linked List is:
-10 -5 0 4 8 23
```