1. Qual a diferença entre malloc e calloc?  
   Malloc é uma função que aloca espaço pra um bloco de bytes na memória RAM e retorna um endereço do tipo void desse bloco de bytes.  
   A Calloc também é uma função para alocação de memória mas recebe um tamanho além de receber os bytes a serem alocados, e retorna um ponteiro void pro primeiro byte que foi alocado, além de limpar o lixo de memória.  
     
   As duas funções retornam um ponteiro null se não tiver memória
2. Qual a diferença entre atribuição implicita e explicita?

Atribuição implícita é um tipo de conversão feito implicitamente pelo compliador, sem precisar de um código ou sintaxe para isso, já a explícita é feita usando a adição de um prefixo “(tipo)” às variáveis que serão convertidas para que sejam convertidas de forma forçada usando o type casting (que seria uma conversão da variável para um tipo específico).

1. Qual a diferença entre heap e stack?  
   Heap é a área de alocação de memória dinâmica, e ele permite que haja desalocação de memória. É a memória global do programa que salva variáveis e dados do programa durante a execução.  
   O Stack também é uma área disponibilizada dentro do espaço de endereçamento mas é estruturado em formato de pilha, quando uma função é chamada no programa ele põe um bloco de memória no topo do stack pra salvar as referências da função e quando termina a execução dela o bloco é desalocado. Aloca dados/variáveis quando uma função é chamada e desaloca quando a função termina.

Olhando a nível de memória, quando eu tenho três linhas:

1. int variavel1 = 10;
2. int \*variavel2 = &variavel1;
3. int \*variavel3 = variavel2;

A nível de referência de memória, descreva o que sabemos sobre as linhas acima.

1. Declara a variavel1 com o tipo int e atribui o valor 10 à mesma
2. Declara a variavel2 com o \* para o compilador saber que ela vai ser um ponteiro que vai guardar o endereço da variavel1
3. Declara a variavel3 que é um ponteiro int e atribui o ponteiro variavel2 a ela.

Todas essas variáveis estão sendo alocadas no heap pois não fazem parte de uma função para serem salvas no stack.

Bubble  
Complexidade temporal pior caso = O(n²)  
Complexidade temporal melhor caso = O(n)  
Complexidade espacial = O(1) uso de 5 variáveis na memória

Insertion   
Complexidade temporal pior caso = O(n²)  
Complexidade temporal melhor caso = O(n)  
Complexidade espacial = O(1) uso de 5 variáveis na memória

Selection  
Complexidade temporal pior caso = O(n²)  
Complexidade temporal melhor caso = O(n²)  
Complexidade espacial = O(1) uso de 5 variáveis na memória  
Complexidade Espacial para os três é sempre constante = O(1)

Merge  
Complexidade temporal pior caso = O(n log(n))  
Complexidade temporal melhor caso = O(n log(n))  
Complexidade Espacial não é constante pois depende do tamanho do array e das suas subdivisões = O(n)

Ponto fraco: usa vetor auxiliar e aumenta o tempo de execução e consumo de memória

Ponto forte: Trabalha bem com quantidade maior de dados   
  
Quick  
Complexidade temporal pior caso = O(n²) ordenar em relação ao pivô depois refazer o processo de forma equivalente em toda a lista.  
Complexidade temporal melhor caso = O(n log(n))  
Complexidade espacial = O(log(n)) uso de 6 variáveis na memória

Melhor caso é quando o menor tempo de execução e memória possíveis foram utilizados, ou seja, o algoritmo teve as condições ideais. e o pior caso é quando o algoritmo tem o seu pior desempenho e geralmente é essa a medida para análise de algoritmo.

Quick sort não é bom para classificar muitos dados por ter muita recursividade e ser complexo, diferente do merge.  
Merge precisa de mais espaço pois tem um array auxiliar, o quick sort utiliza apenas a matriz original e a divide e ordena sem uma auxiliar.  
Merge é mais eficiente que o quick sort