1. No melhor caso do Quicksort, qual elemento deveria ser selecionado como o pivô? Justifique sua resposta.

No melhor caso, o pivô será a mediana do conjunto, já que assim a entrada será dividida em dois conjuntos de tamanhos parecidos para as próximas iterações, e serão necessárias logn iterações para dividir todo o conjunto. Assim, o elemento escolhido para ser o pivô no melhor caso depende dos dados da entrada. No conjunto 1 2 3 4 5 por exemplo, o primeiro pivô a ser escolhido seria o 3, já que é a mediana entre os elementos do conjunto atual.

2. Descreva as estratégias para selecionar o pivô e o pior caso de cada estratégia.

1: Escolher um elemento de posição fixa, como por exemplo o primeiro elemento. Tem como pior caso uma entrada com dados quase ordenados, sendo o pivô o maior ou menor dentre todos eles. Assim, caso o pivô escolhido seja por exemplo o primeiro elemento, o pior caso ocorrerá com os dados completamente ordenados, já que em toda iteração será criado um subconjunto com apenas o pivô e outro com o resto dos elementos.

2: Escolher como pivô a mediana de algum subconjunto da entrada, como por exemplo entre os elementos da primeira posição, do meio, e o último. No pior caso, a maioria dos elementos do subconjunto seriam os maiores ou menores dos dados, fazendo com que a mediana seja sempre um valor próximo dos maiores ou menores elementos, dependendo da quantidade de elementos no subconjunto. Assim, em um subconjunto de tamanho 3 por exemplo, o pior caso ocorreria quando 2 desses elementos sempre fossem os maiores ou menores do subconjunto, pois assim a mediana, que será o pivô, seria o segundo maior ou menor valor, criando um conjunto com os 2 números e outro com todos os restantes para a próxima iteração.

3: Decidir qual elemento será o pivô utilizando um gerador de números aleatórios. No pior caso, o elemento na posição do número gerado é sempre o maior ou menor dos elementos restantes. Assim, no conjunto 2 4 3 1 5, por exemplo, o primeiro índice gerado aleatoriamente seria 3 ou 4, que teriam como pivôs os elementos 1 e 5, respectivamente.

3. Qual é a configuração de entrada que leva ao pior caso do Quicksort? Dê um exemplo de uma configuração de entrada que leva ao pior caso.

Uma entrada que faça com que o pivô escolhido seja o maior ou menor elemento em cada iteração. Um exemplo é a escolha do pivô como o primeiro elemento em um conjunto que já está ordenado, como (1, 2, 3, 4, 5).

4. Por que o Quicksort é considerado o melhor algoritmo de ordenação mesmo possuindo o pior caso igual a $O(n^2)$?

Porque ainda que sua complexidade seja $O(n^2)$, ele possui um desempenho melhor em relação ao caso médio, que possui complexidade O(nlogn).

5. Em quais situações não deveríamos usar o Quicksort?

Nas situações em que possuímos mais informações sobre os dados de entrada, já que será possível escolher um algoritmo mais eficiente para aquele caso, como no caso do insertion sort quando a entrada de dados é pequena ou eles já estão ordenados.

6. Apresente o algoritmo do Quicksort iterativo.

É bem parecido com a versão recursiva, com a diferença de que ao invés de chamar a função novamente passando os índices como parâmetro é utilizada uma pilha para guardar as informações sobre o início e fim dos conjuntos para acessá-los nas próximas iterações.

```
// chamada inicial
public static void quicksort(int[] nums) {
    int esq, dir, pivo;
// iniciar pilha com 0 e tamanho do vetor
    ArrayList <Integer> indices = new ArrayList <Integer> ();
    indices.add(0);
    indices.add(nums.length);
// repetir até a pilha estar vazia
    while ( indices.size() != 0 ) {
// obter indices da iteracao atual e atualizar pilha
        dir = indices.get(indices.size()-1);
        indices.remove(indices.size()-1);
        esq = indices.get(indices.size()-1);
        indices.remove(indices.size()-1);
// particionar caso conjunto tenha mais de 1 elemento
        if ( dir - esq > 1 ) {
// escolher pivo aleatoriamente
            Random gerador = new Random();
            pivo = esq + gerador.nextInt(dir-esq);
// chamada do metodo de particionar
            pivo = particionar ( nums, pivo, esq, dir );
// adicionar indices dos conjuntos das proximas iteracoes na pilha
            indices.add(pivo+1);
            indices.add(dir);
```

```
            indices.add(esq);
            indices.add(pivo);
        }
    }
  }
// metodo para particionar, funciona como no quicksort normal
  public static int particionar ( int [] nums, int indicePivo, int esq, int dir ) {
    int temp, i = esq, j = dir-2;
    int pivo = nums[indicePivo];
    nums[indicePivo] = nums[dir-1];
    nums[dir-1] = pivo;
    while ( i < j ) {
      if ( nums[i] < pivo )
        i++;
      else if ( nums[j] >= pivo )
      j--;
      else {
        temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
      }
    }
    int indice = j;
    if ( nums[j] < pivo ) {
      indice++;
    }
    temp = nums[dir-1];
    nums[dir-1] = nums[indice];
    nums[indice] = temp;
    return indice;
  }
```