1- A lista precisa estar ordenada porque o algoritmo depende da comparação entre o elemento procurado e o valor médio atual para decidir em qual metade da lista continuar. Em uma lista desordenada, essa lógica falha.

2-Casos em que Interpolation Search é mais eficiente:

Listas ordenadas com intervalos uniformes, como {10, 20, 30, 40, 50}.

Em listas grandes onde o elemento procurado está mais próximo dos extremos.

Comparação com Binary Search:

Binary Search: Divide a lista ao meio (O(log n)).

Interpolation Search: Aproxima a posição do elemento diretamente, podendo ser mais rápido em listas uniformes.

3-Tamanho ideal do salto:

O salto ideal é aproximadamente Raiz de n, onde n é o tamanho da lista. Esse valor minimiza o número de comparações no pior caso.

Comparação de Tempo com Binary Search:

Tamanho da Lista	Binary Search (O(log n))	Jump Search (O($\sqrt{}$ n))
100	~7 comparações	~10 comparações
10,000	~14 comparações	~100 comparações

4-O Exponential Search é mais eficiente que o Binary Search quando o elemento está nos primeiros índices, reduzindo o número de comparações iniciais.

5-A sequência de Knuth é mais eficiente em média do que a de Shell, pois reduz melhor o número de comparações.

6-"Dividir para conquistar": Divide a lista em sublistas menores até que cada uma contenha um único elemento. Em seguida, combina as sublistas ordenadas.

10- Último elemento (como na implementação acima):

Simples e comum, mas pode ser ineficiente para listas quase ordenadas.

Primeiro elemento:

Pode ser mais eficiente para listas desordenadas, mas sofre em listas quase ordenadas.

Elemento do meio:

Reduz a chance de pivô mal posicionado, melhorando a eficiência em listas desordenadas.

14-

Algoritmo	Melhor Caso	Caso Médio	Pior Caso	Complexidade de Espaço	Estável?
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$	N/A
Interpolation Search	$O(1)$	$O(\log \log n)$	$O(n)$	$O(1)$	N/A
Shell Sort	$O(n \log n)$	$O(n^{3/2})$	$O(n^2)$	$O(1)$	Não
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Sim
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	Não
Radix Sort	$O(d \cdot n)$	$O(d \cdot n)$	$O(d \cdot n)$	$O(n + k)$	Sim

Exemplo:

Lista: (4,A),(2,B),(2,C),(4,D)(4, A), (2, B), (2, C), (4, D)(4,A),(2,B),(2,C),(4,D)

- **Estável:** (2,B),(2,C),(4,A),(4,D)(2, B), (2, C), (4, A), (4, D)(2,B),(2,C),(4,A),(4,D)
- **Instável:** (2,C),(2,B),(4,D),(4,A)(2, C), (2, B), (4, D), (4, A)(2,C),(2,B),(4,D),(4,A)

18-

Algoritmo	Estável?	Explicação
Bubble Sort	Sim	Mantém a ordem de elementos iguais.
Merge Sort	Sim	Não altera a posição relativa de iguais.
Quick Sort	Não	Pode alterar a posição devido à partição.
Selection Sort	Não	Trocas podem alterar a posição.
Radix Sort	Sim	Baseado em ordenação estável em cada passo.

19-Podemos usar bibliotecas de gráficos (ex.: Gnuplot ou Python) para visualizar como os elementos são reorganizados a cada etapa.