d) Radixsort turbinado em C: a Tabela K.4 mostra um ganho aproximado do RadixsortTurbinado de 1,2 em relação ao RadixsortIntEx em C do Programa J.14, e de 1,5 em relação ao Radixsort em C do Programa D.23. Com relação ao Quicksort em C do Programa D.7, o ganho variou de 3,6 para 10.000 chaves a 4,9 para 100.000.000 de chaves.

Tabela K.4 *Ordem aleatória dos registros com chaves inteiras de 32* bits

	10^4	10^5	10^6	10^7	10^8
RadixsortTurbinado	1	1	1	1	1
RadixsortMelhorado	1,2	1,2	1,2	1,2	1,2
Radixsort	1,3	1,4	1,5	1,5	1,5
Quicksort	3,6	4,2	4,2	4,3	4,9

4.25.

A Tabela K.5 apresenta uma comparação entre o RadixsortCar em C do Programa D.25 e o Quicksort em C do Programa D.7. O compilador Gnu C foi usado com a opção de otimização do código compilado.

Tabela K.5 *Ordem aleatória dos registros com chaves de 1, 2, 4, 8, 12, 16, 20, 24 e 32 caracteres*

	1	2	4	8	12	16	20	24	32
Radixsort	1	1	1	1	1	1	1,6	2,2	3,2
Quicksort	15,2	10,3	6	2,9	1,5	1,1	1	1	1

Capítulo 5

5.1.

a)

	Sequencial	Binária	Hashing
Vantagem	simplicidade	eficiência	eficiência
Desvantagem	custo elevado	arranjo deve estar ordenado	não recupera em ordem alfabética

b)

	Sequencial	Binária	Hashing
Pior caso	$O(n)$	$O(\log n)$	$O(n)$
Caso médio	$O(n)$	$O(\log n)$	$O(1)$

c)

	Sequencial	Binária	Hashing
Memória	boa utilização	boa utilização	$\alpha = \frac{n}{m}$, em geral $\alpha < 80\%$

5.2.

a) Podemos representar o problema por uma **árvore binária de pesquisa** completamente balanceada, na qual o número de nós externos é igual a $n+1$.

Entretanto, o número de nós externos é $\leq 2^h$, onde h corresponde à altura da árvore. Combinando as equações, temos: $n+1 \leq 2^h$, ou $h \geq \log n + 1$, ou $h = \lceil \log n + 1 \rceil$.

b) Ponto importante: existem $n+1$ respostas possíveis. A lista deve ser dividida igualmente em duas sublistas. Repetir o processo em uma das sublistas, até que sobre um elemento. Neste ponto mais uma comparação é necessária para decidir se a chave está presente. Logo, o limite inferior é $h = \lceil \log n + 1 \rceil$.

c) Sim, o Programa 5.3.

5.3. Para cada nó, todos os registros com chaves menores do que a chave que rotula o nó estão na subárvore à esquerda e todos os registros com chaves maiores estão na subárvore à direita.

5.8.

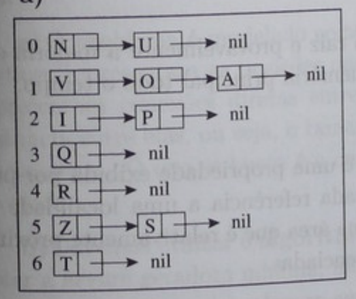
b) O melhor caso ocorre quando os nós estão o mais próximos possíveis da raiz, isto é, quando a árvore binária de pesquisa for uma árvore completa. Neste caso, o número de nós n da árvore será no máximo $2^h - 1$ para a altura (nível) h e no máximo $2^{h-1} - 1$ para a altura $h-1$. Logo, $2^{h-1} - 1 < n \leq 2^h - 1 \Rightarrow 2^{h-1} < n+1 \leq 2^h \Rightarrow h - 1 < \log(n+1) \leq h \Rightarrow h = \lceil \log n + 1 \rceil$.

5.14.

a) A tabela *hash* deve ser utilizada quando o objetivo é ter eficiência nas operações de pesquisa, inserção e remoção, desde que o número de inserções e remoções não provoque variações grandes no valor de n. A tabela *hash* também é indicada quando não há a necessidade de considerar a ordem das chaves e de saber a posição da chave de pesquisa em relação a outras chaves.

5.16. Chaves: N I V O Z A P Q R S T U. Usando $A = 1, B = 2, \cdots, Z = 26$.

a)



b)