Atividade 1 modulo 2

Complexidade de algoritmos

Questão 1

1. struct Elemento{
2. int dado;
3. elemento \*pro;
4. };
5. struct Lista{
6. elemento\* ini;
7. elemento\* fim;
8. int tam;
9. };
10. lista\* create(){
11. lista\* q = (lista\*)malloc(sizeof(lista)); 1
12. q->fim == NULL; 1
13. q->ini == NULL; 1
14. q->tam = 0; 1
15. return q;
16. }
17. elemento\* createElemento(int n){
18. elemento\* num = (elemento\*)malloc(sizeof(elemento)); 1
19. num->dado = n; 1
20. return num;
21. }
22. int vazia(lista\* q){
23. return (q->ini == NULL);
24. }
25. int size(lista\* q){
26. return q->tam;
27. }
28. int isEmpty(lista\* q){
29. if(vazia(q)){ 1
30. return 0; 1
31. } else{ 1
32. return -1; 1
33. }
34. }
35. int add(lista\* q, elemento\* num){
36. if(q->tam == 0){ 1
37. q->ini = num; 1
38. q->fim = num; 1
39. num->pro = NULL; 1
40. } else{ 1
41. q->fim->pro = num; 1
42. q->fim = num; 1
43. num->pro = NULL; 1
44. }
45. q->tam++; 1
46. return 0; 1
47. }
48. int remover(lista\* q){
49. elemento\* aux = (elemento\*)malloc(sizeof(elemento)); 1
50. if(vazia(q)) 1
51. return -1; 1
52. else{
53. aux = q->ini; 1
54. q->ini = aux->pro; 1
55. q->tam--; 1
56. return 0; 1
57. }
58. free(aux); 1
59. }
60. void imprime(lista\* q){
61. elemento\* aux = (elemento\*)malloc(sizeof(elemento)); 1
62. if(vazia(q)) 1
63. printf("Erro: lista vazia!\n"); 1
64. else{ 1
65. aux = q->ini; O(n) 1
66. printf("Lista: "); 1
67. while(aux != NULL){ n
68. printf(" %d", aux->dado); n
69. aux = aux->pro; n
70. }
71. }
72. }
73. int find(lista\* q, elemento\* num){
74. elemento\* aux = (elemento\*)malloc(sizeof(elemento)); 1
75. if(vazia(q)) 1
76. return -1; 1
77. else{ 1
78. aux = q->ini; 1
79. for(int i = 0; i < q->tam; i++){ n
80. if(num->dado == aux->dado) n
81. return i; n
82. aux = aux->pro; O(n) n
83. }
84. return -1; 1
85. }
86. }
87. int clear(lista\* q){
88. if(vazia(q)) 1
89. return -1; 1
90. else{ 1
91. for(int j = q->tam; j > 0; j--){ O(n) n
92. remover(q); n
93. }
94. return 0; 1
95. q->tam = 0; 1
96. }
97. }

Questão 2

1. struct Elemento{
2. int dado;
3. elemento \*ante;
4. elemento \*pro;
5. };
6. struct Lista{
7. elemento\* ini;
8. elemento\* fim;
9. int tam;
10. };
11. lista\* create(){
12. lista\* q = (lista\*)malloc(sizeof(lista)); 1
13. q->fim == NULL; 1
14. q->ini == NULL; 1
15. q->tam = 0; 1
16. return q; 1
17. }
18. elemento\* createElemento(int n){
19. elemento\* num = (elemento\*)malloc(sizeof(elemento)); 1
20. num->dado = n; 1
21. num->ante = NULL; 1
22. num->pro = NULL; 1
23. return num; 1
24. }
25. int vazia(lista\* q){
26. return (q->ini == NULL);
27. }
28. int size(lista\* q){
29. return q->tam;
30. }
31. int isEmpty(lista\* q){
32. if(vazia(q)){ 1
33. return 0; 1
34. } else{ 1
35. return -1; 1
36. }
37. }
38. int add(lista\* q, elemento\* num){
39. if(q->tam == 0){ 1
40. q->ini = num; 1
41. q->fim = num; 1
42. num->pro = NULL; 1
43. } else{ 1
44. elemento \* aux = (elemento\*) malloc(sizeof(elemento)); 1
45. aux = q->fim; 1
46. num->ante = aux; 1
47. aux->pro = num; 1
48. q->fim = num; 1
49. }
50. q->tam++; 1
51. return 0; 1
52. }
53. int remover(lista\* q){
54. elemento\* aux = (elemento\*)malloc(sizeof(elemento)); 1
55. if(vazia(q)) 1
56. return -1; 1
57. else{ 1
58. if(q->tam == 1){ 1
59. aux->ante = NULL; 1
60. aux->pro = NULL; 1
61. aux->dado = NULL; 1
62. q->ini = NULL; 1
63. q->fim = NULL; 1
64. q->tam--; 1
65. return 0; 1
66. }
67. aux = q->ini; 1
68. q->ini = aux->pro; 1
69. q->ini->ante = NULL; 1
70. q->tam--; 1
71. return 0; 1
72. }
73. free(aux); 1
74. }
75. int remove\_(lista\* q,int valor){
76. elemento\* aux1 = (elemento\*)malloc(sizeof(elemento)); 1
77. elemento\* aux2 = (elemento\*)malloc(sizeof(elemento)); 1
78. if(vazia(q)) 1
79. return -1; 1
80. else{ 1
81. aux1 = q->ini; 1
82. if(q->tam == 1){ 1
83. aux1->ante = NULL; 1
84. aux1->pro = NULL; 1
85. aux1->dado = NULL; 1
86. q->ini = NULL; 1
87. q->fim = NULL; 1
88. q->tam--; 1
89. return 0; 1
90. }
91. if(q->ini == q->fim){ 1
92. q->ini = NULL; 1
93. q->fim = NULL; 1
94. }
95. if(q->ini->dado == valor){ 1
96. q->ini = aux1->pro; 1
97. q->ini->ante = NULL; 1
98. q->tam--; 1
99. return 0; 1
100. }
101. while(aux1 != q->fim && aux1->dado != valor){ n
102. aux1 = aux1->pro; n
103. if(valor == aux1->dado){ n
104. aux2 = aux1->ante; n
105. aux2->pro = aux1->pro; n
106. aux2 = aux1->pro->ante; n
107. aux2->ante = aux1->ante->ante; O(n) n
108. q->tam--; n
109. aux1->ante = NULL; n
110. aux1->pro = NULL; n
111. aux1->dado = NULL; n
112. return 0; n
113. }
114. }
115. }
116. }
117. void imprime(lista\* q){
118. elemento\* aux = (elemento\*)malloc(sizeof(elemento)); 1
119. if(vazia(q)) 1
120. printf("Erro: lista vazia!\n"); 1
121. else{ 1
122. int cont = 0; 1
123. aux = q->ini; 1
124. printf("Lista: "); 1
125. while(cont != q->tam){ n
126. printf(" %d", aux->dado); O(n) n
127. aux = aux->pro; n
128. cont ++; n
129. }
130. }
131. }
132. int find(lista\* q, elemento\* num){
133. elemento\* aux = (elemento\*)malloc(sizeof(elemento)); 1
134. if(vazia(q)) 1
135. printf("Erro: lista vazia!\n"); 1
136. else{ 1
137. aux = q->ini; 1
138. for(int i = 0; i < q->tam; i++){ 1
139. if(num->dado == aux->dado){ 1
140. return i; 1
141. }
142. aux = aux->pro; 1
143. }
144. return -1; 1
145. }
146. }
147. int clear(lista\* q){
148. if(vazia(q)) 1
149. return -1; 1
150. else{ 1
151. for(int j = q->tam; j > 0; j--){ 1
152. remover(q); 1
153. }
154. return 0; 1
155. q->tam = 0; 1
156. }
157. }

Questão 3

1. struct Lista{
2. int \*A;
3. int ini;
4. int fim;
5. };
6. int tam;
7. lista \* inicializaLista(){
8. lista \* a = (lista\*) malloc(sizeof(lista));
9. return a;
10. }
11. void create(lista \*a,int n){
12. a->A[n];
13. tam=n; 1
14. printf("Lista criada com sucesso!\n"); 1
15. }
16. void ini(lista \*a){
17. a->fim=0; 1
18. a->ini=0; 1
19. }
20. int vazia(lista \*a){
21. return a->fim == a->ini;
22. }
23. int cheia(lista \*a){
24. return a->fim ==tam;
25. }
26. int isFull(lista \*a){
27. if(cheia(a)){ 1
28. return 0; 1
29. }
30. else { 1
31. return -1; 1
32. }
33. }
34. int isEmpty(lista \*a){
35. if(vazia(a)){ 1
36. return 0; 1
37. }
38. else{ 1
39. return -1; 1
40. }
41. }
42. int size(lista \*a){
43. return (a->fim-a->ini);
44. }
45. int add(lista \*a,int e){
46. if(cheia(a)){ 1
47. return 1; 1
48. printf("Lista cheia!\n"); 1
49. }
50. else{ 1
51. a->A[a->fim] = e; 1
52. if(a->fim != 0){ 1
53. for(int i=a->ini;i<a->fim;i++){ n
54. for(int j=i;j<=a->fim;j++){ n-1
55. if(a->A[i] > a->A[j]){ O(n²) n-1
56. int aux; n-1
57. aux = a->A[j]; n-1
58. a->A[j] = a->A[i]; n-1
59. a->A[i] = aux; n-1
60. }
61. }
62. }
63. }
64. a->fim ++; 1
65. return 0; 1
66. }
67. }
68. int remover(lista \*a,int e){
69. if(vazia(a)){ 1
70. return -1; 1
71. }
72. else{ 1
73. int cont; 1
74. for(int i=a->ini;i<a->fim;i++){ O(n) n
75. if(a->A[i] == e){ n
76. cont = 1; n
77. }
78. if(cont == 1){ n
79. a->A[i]=a->A[i+1]; n
80. }
81. }
82. if(cont == 1){ 1
83. a->fim --; 1
84. return 0; 1
85. }
86. }
87. }
88. void imprime(lista \*a){
89. if(vazia(a)){ 1
90. printf("Erro: lista vazia."); 1
91. }
92. else{ 1
93. for(int i=a->ini;i<a->fim;i++){ O(n) n
94. printf("%d\n",a->A[i]); n
95. }
96. }
97. }
98. int linearSearch(lista \*a,int e){
99. for(int i=a->ini;i<a->fim;i++){ n
100. if(a->A[i]==e){ O(n) n
101. return i; n
102. }
103. }
104. return -1; 1
105. }
106. int bynarySearch(lista \*a,int e){
107. int esquerda=a->ini; 1
108. int direita=a->fim; 1
109. while(esquerda<=direita){ log n
110. int meio = (esquerda+direita)/2; O(log n) log n
111. if(a->A[meio]==e){ log n
112. return meio; log n
113. }
114. else if(a->A[meio]>e){ log n
115. direita=meio - 1; log n
116. }
117. else{ log n
118. esquerda=meio + 1; log n
119. }
120. }
121. return -1; 1
122. }
123. void clear(lista \*a){
124. for(int i=a->ini;i<a->fim;i++){ n
125. a->A[i]=0; O(n) n
126. }
127. a->fim=a->ini; 1
128. printf("Lista vazia!\n"); 1
129. }