Atividade 1 modulo 2

Complexidade de algoritmos

Questão 1

```
1.  struct Elemento{
2.  int dado;
3.  elemento *pro;
4.  };
5.
6.  struct Lista{
7.  elemento* ini;
8.  elemento* fim;
9.  int tam;
10. };
11.
12. lista* create(){
13. lista* q = (lista*)malloc(sizeof(lista));                    1
14. q->fim == NULL;                                              1
15. q->ini == NULL;                                              1
16. q->tam = 0;                                                  1
17. return q;
18. }
19.
20. elemento* createElemento(int n){
21. elemento* num = (elemento*)malloc(sizeof(elemento));          1
22. num->dado = n;                                               1
23. return num;
24. }
25.
26. int vazia(lista* q){
27. return (q->ini == NULL);
28. }
29.
30. int size(lista* q){
31. return q->tam;
32. }
33.
34. int isEmpty(lista* q){
35. if(vazia(q)){                                                1
36. return 0;                                                    1
37. } else{                                                      1
38. return -1;                                                   1
39. }
40. }
41.
42. int add(lista* q, elemento* num){
43. if(q->tam == 0){                                             1
44. q->ini = num;                                                1
```

```
45.     q->fim = num;                                                   1
46.     num->pro = NULL;                                                1
47.     } else{                                                         1
48.     q->fim->pro = num;                                              1
49.     q->fim = num;                                                   1
50.     num->pro = NULL;                                                1
51.     }
52.     q->tam++;                                                       1
53.     return 0;                                                       1
54.     }
55.
56.     int remover(lista* q){
57.     elemento* aux = (elemento*)malloc(sizeof(elemento));            1
58.     if(vazia(q))                                                    1
59.     return -1;                                                      1
60.     else{
61.     aux = q->ini;                                                   1
62.     q->ini = aux->pro;                                              1
63.     q->tam--;                                                       1
64.     return 0;                                                       1
65.     }
66.     free(aux);                                                      1
67.     }
68.
69.     void imprime(lista* q){
70.     elemento* aux = (elemento*)malloc(sizeof(elemento));            1
71.     if(vazia(q))                                                    1
72.     printf("Erro: lista vazia!\n");                                 1
73.     else{                                                           1
74.     aux = q->ini;                          O(n)                     1
75.     printf("Lista: ");                                              1
76.     while(aux != NULL){                                             n
77.     printf(" %d", aux->dado);                                       n
78.     aux = aux->pro;                                                 n
79.     }
80.     }
81.     }
82.
83.     int find(lista* q, elemento* num){
84.     elemento* aux = (elemento*)malloc(sizeof(elemento));            1
85.     if(vazia(q))                                                    1
86.     return -1;                                                      1
87.     else{                                                           1
88.     aux = q->ini;                                                   1
```

```
89.     for(int i = 0; i < q->tam; i++){                          n
90.     if(num->dado ==  aux->dado)                               n
91.     return i;                                                 n
92.     aux = aux->pro;                          O(n)             n
93.     }
94.     return -1;                                                1
95.     }
96.     }
97.
98.     int clear(lista* q){
99.     if(vazia(q))                                              1
100.    return -1;                                                1
101.    else{                                                     1
102.    for(int j = q->tam; j > 0; j--){         O(n)             n
103.    remover(q);                                               n
104.    }
105.    return 0;                                                 1
106.    q->tam = 0;                                               1
107.    }
108.    }
```

Questão 2

```
1.  struct Elemento{
2.  int dado;
3.  elemento *ante;
4.  elemento *pro;
5.  };
6.
7.  struct Lista{
8.  elemento* ini;
9.  elemento* fim;
10. int tam;
11. };
12.
13. lista* create(){
14. lista* q = (lista*)malloc(sizeof(lista));                    1
15. q->fim == NULL;                                              1
16. q->ini == NULL;                                              1
17. q->tam = 0;                                                  1
18. return q;                                                    1
19. }
20.
21. elemento* createElemento(int n){
22. elemento* num = (elemento*)malloc(sizeof(elemento));         1
23. num->dado = n;                                               1
24. num->ante = NULL;                                            1
25. num->pro = NULL;                                             1
26. return num;                                                  1
27. }
28.
29. int vazia(lista* q){
30. return (q->ini == NULL);
31. }
32.
33. int size(lista* q){
34. return q->tam;
35. }
36.
37. int isEmpty(lista* q){
38. if(vazia(q)){                                                1
39. return 0;                                                    1
40. } else{                                                      1
41. return -1;                                                   1
42. }
43. }
44.
```

```
45.     int add(lista* q, elemento* num){
46.     if(q->tam == 0){                                                    1
47.     q->ini = num;                                                       1
48.     q->fim = num;                                                       1
49.     num->pro = NULL;                                                    1
50.     } else{                                                             1
51.     elemento * aux = (elemento*) malloc(sizeof(elemento));              1
52.     aux = q->fim;                                                       1
53.     num->ante = aux;                                                    1
54.     aux->pro = num;                                                     1
55.     q->fim = num;                                                       1
56.     }
57.     q->tam++;                                                           1
58.     return 0;                                                           1
59.     }
60.
61.     int remover(lista* q){
62.     elemento* aux = (elemento*)malloc(sizeof(elemento));                1
63.     if(vazia(q))                                                        1
64.     return -1;                                                          1
65.     else{                                                               1
66.     if(q->tam == 1){                                                    1
67.     aux->ante = NULL;                                                   1
68.     aux->pro = NULL;                                                    1
69.     aux->dado = NULL;                                                   1
70.     q->ini = NULL;                                                      1
71.     q->fim = NULL;                                                      1
72.     q->tam--;                                                           1
73.     return 0;                                                           1
74.     }
75.     aux = q->ini;                                                       1
76.     q->ini = aux->pro;                                                  1
77.     q->ini->ante = NULL;                                                1
78.     q->tam--;                                                           1
79.     return 0;                                                           1
80.     }
81.     free(aux);                                                          1
82.     }
83.
84.     int remove_(lista* q,int valor){
85.     elemento* aux1 = (elemento*)malloc(sizeof(elemento));               1
86.     elemento* aux2 = (elemento*)malloc(sizeof(elemento));               1
87.     if(vazia(q))                                                        1
88.     return -1;                                                          1
```

```
89.     else{                                                   1
90.     aux1 = q->ini;                                          1
91.     if(q->tam == 1){                                        1
92.     aux1->ante = NULL;                                      1
93.     aux1->pro = NULL;                                       1
94.     aux1->dado = NULL;                                      1
95.     q->ini = NULL;                                          1
96.     q->fim = NULL;                                          1
97.     q->tam--;                                               1
98.     return 0;                                               1
99.     }
100.    if(q->ini == q->fim){                                   1
101.    q->ini = NULL;                                          1
102.    q->fim = NULL;                                          1
103.    }
104.    if(q->ini->dado == valor){                              1
105.    q->ini = aux1->pro;                                     1
106.    q->ini->ante = NULL;                                    1
107.    q->tam--;                                               1
108.    return 0;                                               1
109.    }
110.    while(aux1 != q->fim && aux1->dado != valor){           n
111.    aux1 = aux1->pro;                                       n
112.    if(valor == aux1->dado){                                n
113.    aux2 = aux1->ante;                                      n
114.    aux2->pro = aux1->pro;                                  n
115.    aux2 = aux1->pro->ante;                                 n
116.    aux2->ante = aux1->ante->ante;          O(n)            n
117.    q->tam--;                                               n
118.    aux1->ante = NULL;                                      n
119.    aux1->pro = NULL;                                       n
120.    aux1->dado = NULL;                                      n
121.    return 0;                                               n
122.    }
123.    }
124.    }
125.    }
126.
127.    void imprime(lista* q){
128.    elemento* aux = (elemento*)malloc(sizeof(elemento));    1
129.    if(vazia(q))                                            1
130.    printf("Erro: lista vazia!\n");                         1
131.    else{                                                   1
132.    int cont = 0;                                           1
```

```
133.    aux = q->ini;                                                    1
134.    printf("Lista: ");                                               1
135.    while(cont != q->tam){                                           n
136.    printf(" %d", aux->dado);              O(n)                      n
137.    aux = aux->pro;                                                  n
138.    cont ++;                                                         n
139.    }
140.    }
141.    }
142.
143.    int find(lista* q, elemento* num){
144.    elemento* aux = (elemento*)malloc(sizeof(elemento));             1
145.    if(vazia(q))                                                     1
146.    printf("Erro: lista vazia!\n");                                  1
147.    else{                                                            1
148.    aux = q->ini;                                                    1
149.    for(int i = 0; i < q->tam; i++){                                 1
150.    if(num->dado == aux->dado){                                      1
151.    return i;                                                        1
152.    }
153.    aux = aux->pro;                                                  1
154.    }
155.    return -1;                                                       1
156.    }
157.    }
158.
159.    int clear(lista* q){
160.    if(vazia(q))                                                     1
161.    return -1;                                                       1
162.    else{                                                            1
163.    for(int j = q->tam; j > 0; j--){                                 1
164.    remover(q);                                                      1
165.    }
166.    return 0;                                                        1
167.    q->tam = 0;                                                      1
168.    }
169.    }
```

Questão 3

```
1.  struct Lista{
2.  int *A;
3.  int ini;
4.  int fim;
5.  };
6.
7.  int tam;
8.
9.  lista * inicializaLista(){
10. lista * a = (lista*) malloc(sizeof(lista));
11. return a;
12. }
13.
14. void create(lista *a,int n){
15. a->A[n];
16. tam=n;                                                    1
17. printf("Lista criada com sucesso!\n");                    1
18. }
19.
20. void ini(lista *a){
21. a->fim=0;                                                 1
22. a->ini=0;                                                 1
23. }
24.
25. int vazia(lista *a){
26. return a->fim == a->ini;
27. }
28.
29. int cheia(lista *a){
30. return a->fim ==tam;
31. }
32.
33. int isFull(lista *a){
34. if(cheia(a)){                                             1
35. return 0;                                                 1
36. }
37. else {                                                    1
38. return -1;                                                1
39. }
40. }
41.
42. int isEmpty(lista *a){
43. if(vazia(a)){                                             1
44. return 0;                                                 1
```

```
45.     }
46.     else{                                                   1
47.     return -1;                                              1
48.     }
49.     }
50.
51.     int size(lista *a){
52.     return (a->fim-a->ini);
53.     }
54.
55.     int add(lista *a,int e){
56.     if(cheia(a)){                                           1
57.     return 1;                                               1
58.     printf("Lista cheia!\n");                               1
59.     }
60.     else{                                                   1
61.     a->A[a->fim] = e;                                       1
62.
63.     if(a->fim != 0){                                        1
64.     for(int i=a->ini;i<a->fim;i++){                         n
65.     for(int j=i;j<=a->fim;j++){                             n-1
66.     if(a->A[i] > a->A[j]){          O(n²)                   n-1
67.     int aux;                                                n-1
68.     aux = a->A[j];                                          n-1
69.     a->A[j] = a->A[i];                                      n-1
70.     a->A[i] = aux;                                          n-1
71.     }
72.     }
73.     }
74.     }
75.     a->fim ++;                                              1
76.     return 0;                                               1
77.     }
78.     }
79.
80.     int remover(lista *a,int e){
81.     if(vazia(a)){                                           1
82.     return -1;                                              1
83.     }
84.     else{                                                   1
85.     int cont;                                               1
86.     for(int i=a->ini;i<a->fim;i++){         O(n)            n
87.     if(a->A[i] == e){                                       n
88.     cont = 1;                                               n
```

```
89.         }
90.         if(cont == 1){                                                  n
91.         a->A[i]=a->A[i+1];                                              n
92.         }
93.         }
94.         if(cont == 1){                                                  1
95.         a->fim --;                                                      1
96.         return 0;                                                       1
97.         }
98.         }
99.         }
100.
101.        void imprime(lista *a){
102.           if(vazia(a)){                                                1
103.              printf("Erro: lista vazia.");                             1
104.           }
105.           else{                                                        1
106.              for(int i=a->ini;i<a->fim;i++){              O(n)        n
107.              printf("%d\n",a->A[i]);                                   n
108.           }
109.           }
110.        }
111.
112.        int linearSearch(lista *a,int e){
113.           for(int i=a->ini;i<a->fim;i++){                              n
114.              if(a->A[i]==e){                              O(n)        n
115.                 return i;                                             n
116.              }
117.           }
118.           return -1;                                                   1
119.
120.        }
121.
122.        int bynarySearch(lista *a,int e){
123.           int esquerda=a->ini;                                         1
124.           int direita=a->fim;                                          1
125.
126.           while(esquerda<=direita){                                    log n
127.              int meio = (esquerda+direita)/2;          O(log n)        log n
128.              if(a->A[meio]==e){                                        log n
129.                 return meio;                                          log n
130.              }
131.              else if(a->A[meio]>e){                                    log n
132.                 direita=meio - 1;                                     log n
```

```
133.            }
134.            else{                                                   log n
135.                esquerda=meio + 1;                                  log n
136.            }
137.        }
138.        return -1;                                                  1
139.    }
140.
141.    void clear(lista *a){
142.        for(int i=a->ini;i<a->fim;i++){                             n
143.            a->A[i]=0;                              O(n)            n
144.        }
145.        a->fim=a->ini;                                              1
146.        printf("Lista vazia!\n");                                   1
147.    }
```