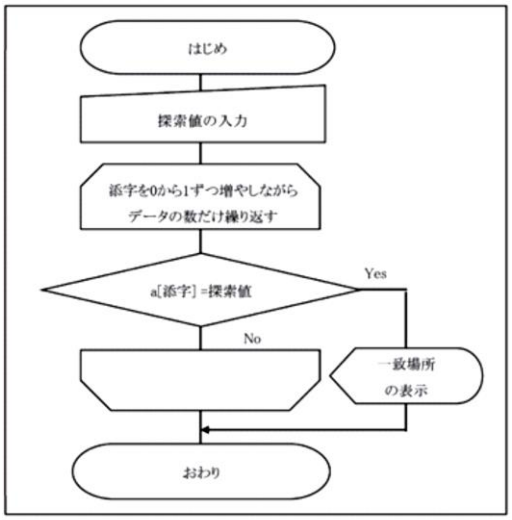
エキスパート活動　課題 A　「線形探索」

線形探索はリストを先頭から順に比較しながら探索値に一致するデータを探し出す探索方法である。探索値とリスト内のデータを先頭から順番に比較していき，一致したデータがあれば，その場所（リストの添字）を表示する。図表 1 のような 7 個のデータ a[0]～a[6]があり，探索値が「82」であるとする。フローチャートでは下記の①～③のように，図表 1 の左側から順に探索値の「82」と比較していくことになる。

リストの要素	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
データ	61	15	82	77	21	32	53

図表 1　探索対象のリスト



図表 2　線形探索

① a[0]の「61」と探索値「82」を比較し，異なるので次に進む。
② a[1]の「15」と探索値「82」を比較し，異なるので次に進む。
③ a[2]の「82」と探索値「82」を比較し，一致するので a[2]にあることがわかる。

＜プログラム例＞

```
data = [1, 5, 3, 8, 2]          # 検索対象のデータリストを定義
target = 8                      # 探したい値を定義
found = False                   # 見つかったかどうかを示すフラグを初期化
index = -1                      # 見つかった要素のインデックスを初期化

for i in range(len(data)):      # リストの各要素を順番に調べていく
  if data[i] == target:         # 現在の要素が探している値（target）と一致するかチェック
    index = i                   # 一致した場合、そのインデックスを記録
    found = True                # 見つかったことを示すフラグを True に設定
    break                       # 見つかったのでループを終了

if found:                       # 探索結果に基づいてメッセージを表示
  print(f"{target} は {index} 番目にあります。")  # 見つかった場合
else:
  print(f"{target} は見つかりませんでした。")      # 見つからなかった場合
```

発問①.　上記のプログラムにおいて、リストの要素を[61,15,82,77,21,32,53]とし、また探索する数値を 82 として実行してみよう。

発問②.　発問①の探索回数は何回になりますか。　答え [　　] 回

発問③.　実行したときに、『探索した回数を表示』できるようにしよう。

エキスパート活動　課題Ｂ　「二分探索」

リストの中から探索範囲を半分ずつ狭めながら目的のデータを探し出す探索方法である。
ここでは７個のデータ a[0]～a[6]があり昇順にソートされており，探索値が「43」であるとする。

リストの要素	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
データ	25	33	43	51	66	71	88

図表４　探索対象のリスト

①中央値＞探索値より，上限添字の入れ替え

まず，探索範囲の中央を求める。リストの下限は a[0]，上限は a[6]なので，中央は「(下限添字＋上限添字）÷２＝(0+6)÷2=3」のように求められる。従って，探索範囲の中央にある a[3]のデータ「51」と探索値「43」を比較する。

「43」は a[3]の「51」より小さいので，探索範囲は a[0]～a[2]に絞られる。

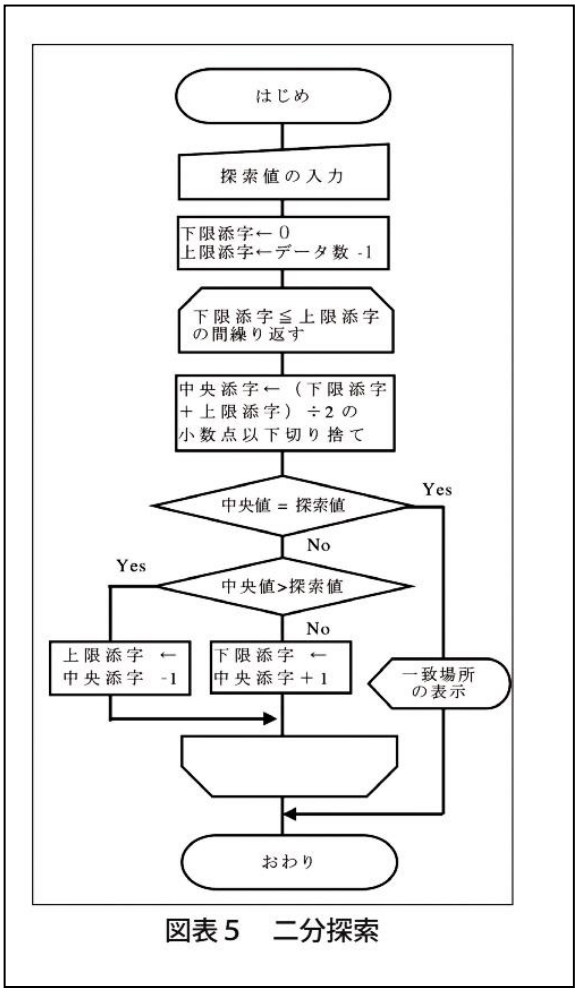
②中央値＜探索値より，下限添字の入れ替え

新たな探索範囲の下限は a[0]，上限は a[2]なので中央は「(0+2)÷2=1」のように求められる。従って，a[1]のデータ「33」と探索値「43」を比較する。

「43」は a[1]の「33」より大きいので，探索範囲は a[2]に絞られるため，下限添字を中央添字の「1」を１つ増やした値の「2」に変更する。

③中央値＝探索値より，検索終了

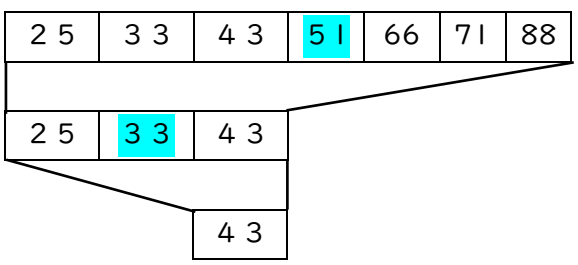
新たな探索範囲の下限は a[2]，上限は a[2]なので中央は「(2+2)÷2=2」である。a[2]のデータ「43」と「43」を比較する。「43」は a[2]の「43」と一致するので，「43」が a[2]にあることが分かる。



図表５　二分探索

※人間であれば，下限が a[2]で上限が a[2]であれば，探索値が a[2]であることは当たり前に感じる。しかし，コンピュータは，定められたアルゴリズムによって動くため，下限と上限が一致するということは中央値もそれと一致するということにしかならない。アルゴリズムに従って，次のステップで中央値と探索値が一致して，探索が完了する。

＜探索のイメージ図＞



＜プログラム例＞

```
arr = [2, 5, 7, 8, 11, 12, 13, 14]        # 探索対象のソート済み配列を定義します
target = 12                                  # 探索するターゲット値を定義します

left = 0                                     # 探索範囲の左端を初期化します
right = len(arr) - 1                         # 探索範囲の右端を初期化します
found_index = -1            # 見つかったインデックスを初期化します。見つからなかった場合は-1 のままです

while left <= right:                         # 左端が右端以下の間、探索を続けます
    mid = (left + right) // 2                # 中央のインデックスを計算します

    if arr[mid] == target:                   # 中央の値がターゲットと一致するか確認します
        found_index = mid                    # 一致した場合、そのインデックスを記録します
        break                                # 探索を終了します
    elif arr[mid] < target:                  # 中央の値がターゲットより小さい場合
        left = mid + 1                       # 探索範囲の左端を中央の次へ移動します
    else:                                    # 中央の値がターゲットより大きい場合
        right = mid - 1                      # 探索範囲の右端を中央の前へ移動します

if found_index != -1:                        # 見つかったインデックスが初期値でなければ
    print(f"要素 {target} はインデックス {found_index} にあります。") # 結果を出力します
else:                                        # 見つかったインデックスが初期値のままであれば
    print(f"要素 {target} は見つかりませんでした。")                  # 結果を出力します
```

発問①.　上記のプログラムにおいて、リストの要素を[25,33,43,51,66,71,88]とし、また探索する数値を 43 として実行してみよう。

発問②.　発問①の探索回数は何回になりますか。　答え [　　] 回

発問③.　実行したときに、『探索した回数を表示』できるようにしよう。

エキスパート活動　課題 C　「線形探索 vs 二分探索」

線形探索と二分探索の比較

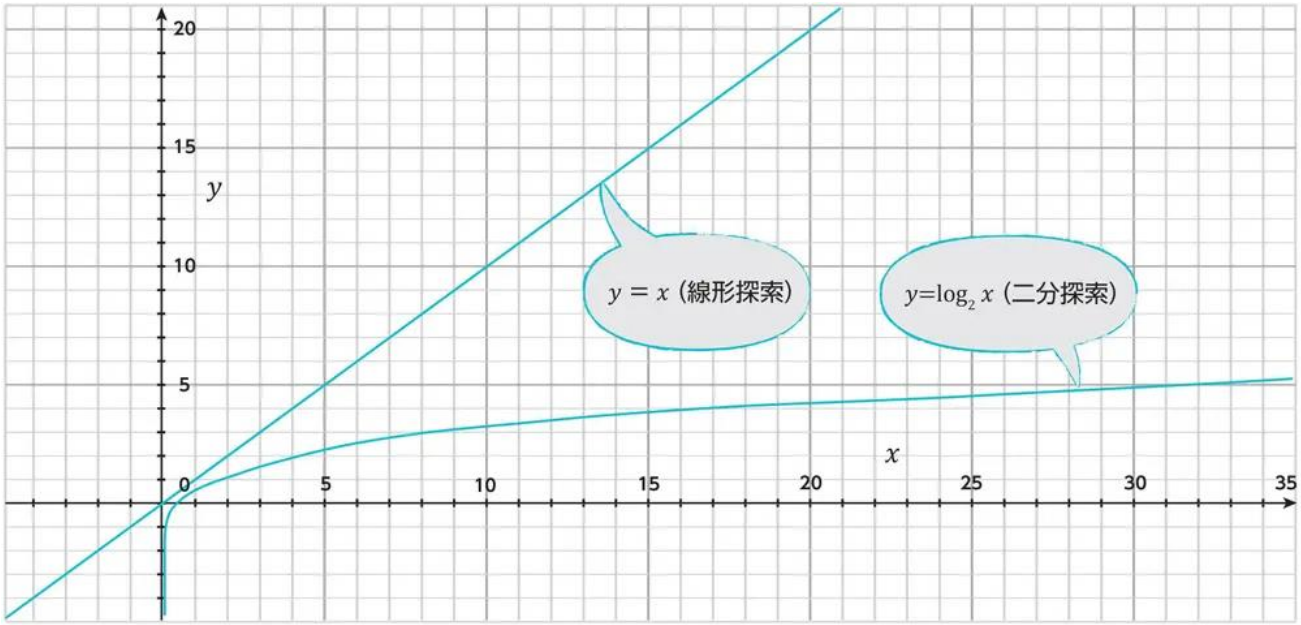
左側から探索を行う線形探索においては探索値が左端にあるか，右端にあるかによって探索回数が大きく変わり，1 回の探索で見つかる場合もあれば，全てのデータを探索する場合も出てくるので，一般的にデータ数が多い場合は線形探索では時間がかかることが多い。

二分探索の場合は 1 回の探索で見つからなかった場合でも探索範囲をデータ数の半分に絞ることができるので，線形探索と比較すると，データ数が大きくなっても探索にかかる時間はそれほど増加しない場合が多い。

最大探索回数だけを比較すると，回数の少ない二分探索がよいアルゴリズムと考えがちだが，二分探索には事前にデータを並べ替えておく必要があり，一概によいアルゴリズムとは言い切れない。例えば「事前にデータが並び替えられている保証がない場合」や「データの数がそれほど多くなく，シンプルなアルゴリズムの方が望ましい場合」などは線形探索の方が有用な場合もある。

データが増えたときの比較回数を考える

二分探索の場合、比較回数の増え方は対数(2 年時の数Ⅱ「指数・対数」で学習予定)のペースであるため、データ数が 1,000 個程度に増えても比較回数は 10 回程度、100 万個に増えても比較回数は 20 回程度です。線形探索だと 1,000 個になると 1,000 回、100 万個になると 100 万回かかっていたものと比べると下図のように圧倒的な差が生まれることがわかります。



一般的には、線形探索よりも二分探索のほうが高速に処理できますが、データが昇順か降順に並んでいる必要があるため、事前にデータの並べ替えが必要です（線形探索の場合には並び順は関係ない）。また、データの個数が少ない場合には処理速度に大きな差が出ないことから、線形探索が使われることも少なくありません。

<プログラム例>

```
# 線形探索と二分探索の探索回数の比較
import random
import matplotlib.pyplot as plt
!pip install -q japanize-matplotlib # 日本語表示のために追加 (-q で出力を抑制)
import japanize_matplotlib # 日本語表示のために追加

# メインの比較ロジック（関数を使用しない）
data_sizes = [10, 50, 100, 500, 1000]  # データサイズを変更
```

```
linear_counts = []
binary_counts = []

for size in data_sizes:
    # データ生成とソート
    data = sorted([random.randint(1, 1000) for _ in range(size)])
    target = random.choice(data) # ターゲットはデータ内に存在する値から選択

    # 線形探索
    linear_search_count = 0
    for i in range(len(data)):
        linear_search_count += 1
        if data[i] == target:
            break
    linear_counts.append(linear_search_count)

    # 二分探索
    binary_search_count = 0
    low = 0
    high = len(data) - 1
    while low <= high:
        binary_search_count += 1
        mid = (low + high) // 2
        if data[mid] == target:
            break
        elif data[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    binary_counts.append(binary_search_count)

# 結果のプロット
plt.plot(data_sizes, linear_counts, label="線形探索", marker='o') # マーカーを追加
plt.plot(data_sizes, binary_counts, label="二分探索", marker='o') # マーカーを追加
plt.xlabel("データサイズ")
plt.ylabel("探索回数")
plt.title("線形探索と二分探索の比較")
plt.legend()
plt.show()
```

発問①.　実行を何度か繰り返した際に、結果からどのようなことが分かるか？特徴と傾向を見つけてみよう。
発問②.　データサイズを変更した場合に結果にどう影響するのか考えてみよう。
発問③.　それぞれのアルゴリズムの長所と短所をまとめよう。