# Advanced Sorting: implementing reverse sort

In this lesson, you'll continue to implement our advanced data sorting technique. By the end of this lesson, we will be able to sort and reverse sort our app's table by column

In the previous lesson, we did some initial ground work to get our search functionality working. In this chapter, we'll finish the sort functionality with reverse sort. The list should perform a reverse sort once you click a Sort component twice. First, you need to define the reverse state with a boolean. The sort can be either reversed or non-reversed.

```
this.state = {
  results: null,
  searchKey: '',
  searchTerm: DEFAULT_QUERY,
  error: null,
  isLoading: false,
  sortKey: 'NONE',
  isSortReverse: false,
};
```

Now in your sort method, you can evaluate if the list is reverse sorted. It is reverse if the `sortKey` in the state is the same as the incoming `sortKey` and the reverse state is not already set to true.

```
onSort(sortKey) {
  const isSortReverse = this.state.sortKey === sortKey && !this.state.isSortReverse;
  this.setState({ sortKey, isSortReverse });
}
```

Again, we pass the reverse prop to your Table component:

```
class App extends Component {

  ...

  render() {
    const {
      searchTerm,
      results,
```

```
      searchKey,
      error,
      isLoading,

      sortKey,
      isSortReverse
    } = this.state;

    ...

    return (
      <div className="page">
        ...
        <Table
          list={list}
          sortKey={sortKey}
          isSortReverse={isSortReverse}
          onSort={this.onSort}
          onDismiss={this.onDismiss}
        />
        ...
      </div>
    );
  }
}
```

The Table has to have an arrow function block body to compute the data now:

```
const Table = ({
  list,
  sortKey,
  isSortReverse,
  onSort,
  onDismiss
}) => {
  const sortedList = SORTS[sortKey](list);
  const reverseSortedList = isSortReverse
    ? sortedList.reverse()
    : sortedList;

  return(
    <div className="table">
      <div className="table-header">
        ...
      </div>
      {reverseSortedList.map(item =>
        ...
      )}
    </div>
  );
}
```

Finally, we want to give the user visual feedback to distinguish which column is actively sorted. Each Sort component has its specific `sortKey` already, which can be used to identify the activated sort. We pass the `sortKey` from the internal component state as active sort key to your Sort component:

```
const Table = ({
  list,
  sortKey,
  isSortReverse,
  onSort,
  onDismiss
}) => {
  const sortedList = SORTS[sortKey](list);
  const reverseSortedList = isSortReverse
    ? sortedList.reverse()
    : sortedList;

  return(
    <div className="table">
      <div className="table-header">
        <span style={{ width: '40%' }}>
          <Sort
            sortKey={'TITLE'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Title
          </Sort>
        </span>
        <span style={{ width: '30%' }}>
          <Sort
            sortKey={'AUTHOR'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Author
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'COMMENTS'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Comments
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'POINTS'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Points
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          Archive
        </span>
      </div>
      {reverseSortedList.map(item =>
          ...
      )}
```

```
    </div>
  );
}
```

Now the user will know whether sort is active based on the `sortKey` and `activeSortKey` . Give your Sort component an extra `className` attribute, in case it is sorted, to give visual feedback:

```
const Sort = ({
  sortKey,
  activeSortKey,
  onSort,
  children
}) => {
  const sortClass = ['button-inline'];

  if (sortKey === activeSortKey) {
    sortClass.push('button-active');
  }

  return (
    <Button
      onClick={() => onSort(sortKey)}
      className={sortClass.join(' ')}
    >
      {children}
    </Button>
  );
}
```

We can define `sortClass` more efficiently using a library called classnames, which is installed using npm:

```
npm install classnames
```

After installation, we import it on top of the *src/App.js* file.

```
import React, { Component } from 'react';
import fetch from 'isomorphic-fetch';
import { sortBy } from 'lodash';
import classNames from 'classnames';
import './App.css';
```

Now we can define `className` with conditional classes:

```
const Sort = ({
  sortKey,
  activeSortKey,
  onSort,
  children
```

```
}) => {
  const sortClass = classNames(
    'button-inline',

    { 'button-active': sortKey === activeSortKey }
  );

  return (
    <Button
      onClick={() => onSort(sortKey)}
      className={sortClass}
    >
      {children}
    </Button>
  );
}
```

There will be failing snapshot and unit tests for the Table component. Since we intentionally changed again our component representations, we accept the snapshot tests, but we still need to fix the unit test. In *src/App.test.js* , provide a `sortKey` and the `isSortReverse` boolean for the Table component.

```
...

describe('Table', () => {

  const props = {
    list: [
      { title: '1', author: '1', num_comments: 1, points: 2, objectID: 'y' },
      { title: '2', author: '2', num_comments: 1, points: 2, objectID: 'z' },
    ],
    sortKey: 'TITLE',
    isSortReverse: false,
  };

  ...

});
```

We may need to accept the failing snapshot tests again for the Table component, because we provided extended props for it. The advanced sort interaction is finally complete.

## App with Table sorting:

```
import React, { Component } from 'react';
import { sortBy } from 'lodash';
import classNames from 'classnames';
require('./App.css');

const DEFAULT_QUERY = 'redux';
const DEFAULT_HPP = '100';
```

```
const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';

const SORTS = {
  NONE: list => list,
  TITLE: list => sortBy(list, 'title'),
  AUTHOR: list => sortBy(list, 'author'),
  COMMENTS: list => sortBy(list, 'num_comments').reverse(),
  POINTS: list => sortBy(list, 'points').reverse(),
};

class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      results: null,
      searchKey: '',
      searchTerm: DEFAULT_QUERY,
      error: null,
      isLoading: false,
      sortKey: 'NONE',
      isSortReverse: false,
    };

    this.needsToSearchTopstories = this.needsToSearchTopstories.bind(this);
    this.setSearchTopstories = this.setSearchTopstories.bind(this);
    this.fetchSearchTopstories = this.fetchSearchTopstories.bind(this);
    this.onSearchChange = this.onSearchChange.bind(this);
    this.onSearchSubmit = this.onSearchSubmit.bind(this);
    this.onDismiss = this.onDismiss.bind(this);
    this.onSort = this.onSort.bind(this);
  }

  onSort(sortKey) {
    const isSortReverse = this.state.sortKey === sortKey && !this.state.isSortReverse;
    this.setState({ sortKey, isSortReverse });
  }

  needsToSearchTopstories(searchTerm) {
    return !this.state.results[searchTerm];
  }

  setSearchTopstories(result) {
    const { hits, page } = result;
    const { searchKey, results } = this.state;

    const oldHits = results && results[searchKey]
      ? results[searchKey].hits
      : [];

    const updatedHits = [
      ...oldHits,
      ...hits
    ];

    this.setState({
      results: {
        ...results,
```

```
      [searchKey]: { hits: updatedHits, page }
    },
    isLoading: false
  });
}

fetchSearchTopstories(searchTerm, page = 0) {
  this.setState({ isLoading: true });

  fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}`)
    .then(response => response.json())
    .then(result => this.setSearchTopstories(result))
      .catch(e => this.setState({ error: e }));
}

componentDidMount() {
  const { searchTerm } = this.state;
  this.setState({ searchKey: searchTerm });
  this.fetchSearchTopstories(searchTerm);
}

onSearchChange(event) {
  this.setState({ searchTerm: event.target.value });
}

onSearchSubmit(event) {
  const { searchTerm } = this.state;
  this.setState({ searchKey: searchTerm });

  if (this.needsToSearchTopstories(searchTerm)) {
    this.fetchSearchTopstories(searchTerm);
  }

  event.preventDefault();
}

onDismiss(id) {
  const { searchKey, results } = this.state;
  const { hits, page } = results[searchKey];

  const isNotId = item => item.objectID !== id;
  const updatedHits = hits.filter(isNotId);

  this.setState({
    results: {
      ...results,
      [searchKey]: { hits: updatedHits, page }
    }
  });
}

render() {
  const {
    searchTerm,
    results,
    searchKey,
    error,
    isLoading,
    sortKey,
    isSortReverse
  } = this.state;
```

```
    const page = (
      results &&
      results[searchKey] &&
      results[searchKey].page
    ) || 0;

    const list = (
      results &&
      results[searchKey] &&
      results[searchKey].hits
    ) || [];

    return (
      <div className="page">
        <div className="interactions">
          <Search
            value={searchTerm}
            onChange={this.onSearchChange}
            onSubmit={this.onSearchSubmit}
          >
            Search
          </Search>
        </div>
        { error
          ? <div className="interactions">
            <p>Something went wrong.</p>
          </div>
          : <Table
                list={list}
                  sortKey={sortKey}
                isSortReverse={isSortReverse}
              onSort={this.onSort}
            onDismiss={this.onDismiss}
                />
        }
        <div className="interactions">
          <ButtonWithLoading
            isLoading={isLoading}
            onClick={() => this.fetchSearchTopstories(searchKey, page + 1)}>
            More
          </ButtonWithLoading>
        </div>
      </div>
    );
  }
}

const Search = ({
  value,
  onChange,
  onSubmit,
  children
}) =>
  <form onSubmit={onSubmit}>
    <input
      type="text"
      value={value}
      onChange={onChange}
    />
    <button type="submit">
      {children}
    </button>
```

```
  </form>

const Table = ({
  list,
  sortKey,
  isSortReverse,
  onSort,
  onDismiss
}) => {
  const sortedList = SORTS[sortKey](list);
  const reverseSortedList = isSortReverse
    ? sortedList.reverse()
    : sortedList;

  return(
    <div className="table">
      <div className="table-header">
        <span style={{ width: '40%' }}>
          <Sort
            sortKey={'TITLE'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Title
          </Sort>
        </span>
        <span style={{ width: '30%' }}>
          <Sort
            sortKey={'AUTHOR'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Author
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'COMMENTS'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Comments
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'POINTS'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Points
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          Archive
        </span>
      </div>
      { reverseSortedList.map(item =>
        <div key={item.objectID} className="table-row">
          <span style={{ width: '40%' }}>
            <a href={item.url}>{item.title}</a>
          </span>
```

```
          <span style={{ width: '30%' }}>
            {item.author}
          </span>
          <span style={{ width: '10%' }}>
            {item.num_comments}
          </span>
          <span style={{ width: '10%' }}>
            {item.points}
          </span>
          <span style={{ width: '10%' }}>
            <Button
              onClick={() => onDismiss(item.objectID)}
              className="button-inline"
            >
              Dismiss
            </Button>
          </span>
        </div>
      )}
    </div>
  );
}

const Button = ({ onClick, className = '', children }) =>
  <button
    onClick={onClick}
    className={className}
    type="button"
  >
    {children}
  </button>

const Loading = () =>
  <div>Loading ...</div>

const withLoading = (Component) => ({ isLoading, ...rest }) =>
  isLoading ? <Loading /> : <Component { ...rest } />

const ButtonWithLoading = withLoading(Button);

const Sort = ({
  sortKey,
  activeSortKey,
  onSort,
  children
}) => {
  const sortClass = classNames(
    'button-inline',
    { 'button-active': sortKey === activeSortKey }
  );

  return (
    <Button
      onClick={() => onSort(sortKey)}
      className={sortClass}
    >
      {children}
    </Button>
  );
}

export default App;
```

```
export {
  Button,
  Search,
  Table,
};
```

## Exercises:

- Use a library like Font Awesome to indicate the (reverse) sort. It could be an arrow up or arrow down icon next to each Sort header

## Further Reading

- Read about the classnames library